Chapter 11

Approximation Algorithms

P and NP

P:

The family of problems that can be solved quickly in polynomial time.

NP:

The family of problems for which a given solution to the problem can be verified in polynomial time.

Remark. NP stands for nondeterministic polynomial-time.

Certifiers and Certificates: 3-Satisfiability

SAT. Given a CNF formula Φ, is there a satisfying assignment?

Certificate. An assignment of truth values to the n boolean variables.

Certifier. Check that each clause in Φ has at least one true literal.

Ex.

$$\left(\overline{x_1} \vee x_2 \vee x_3 \right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3 \right) \wedge \left(x_1 \vee x_2 \vee x_4 \right) \wedge \left(\overline{x_1} \vee \overline{x_3} \vee \overline{x_4} \right)$$

instance s

$$x_1 = 1, \ x_2 = 1, \ x_3 = 0, \ x_4 = 1$$

certificate t

Conclusion. SAT is in NP.

Certifiers and Certificates: Hamiltonian Cycle

HAM-CYCLE. Given an undirected graph G = (V, E), does there exist a simple cycle C that visits every node?

Certificate. A permutation of the n nodes.

Certifier. Check that the permutation contains each node in V exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

Conclusion. HAM-CYCLE is in NP.

instance s

certificate t
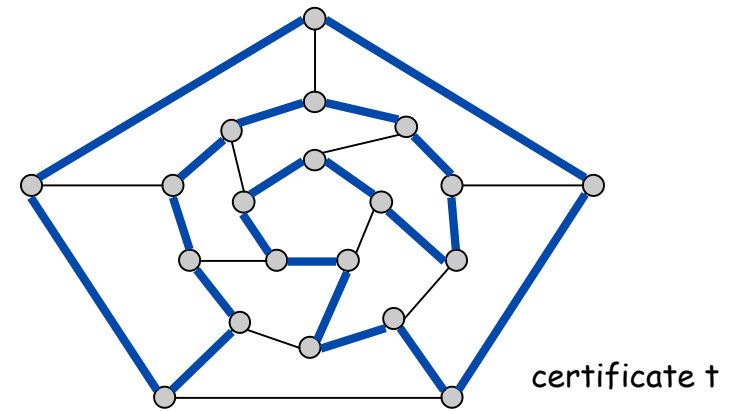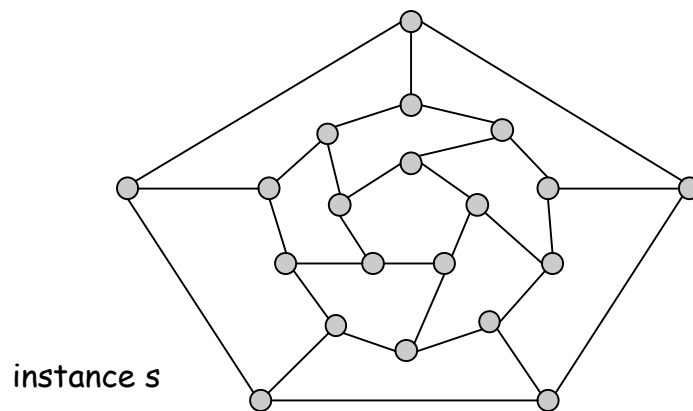
P, NP, EXP

P. Decision problems for which there is a poly-time algorithm.

EXP. Decision problems for which there is an exponential-time algorithm.

NP. Decision problems for which there is a poly-time certifier.

Claim. P $\subseteq$ NP.

Pf. Consider any problem X in P.

- By definition, there exists a poly-time algorithm A(s) that solves X.
- Certificate: t = ε, certifier C(s, t) = A(s). ∎

Claim. NP $\subseteq$ EXP.

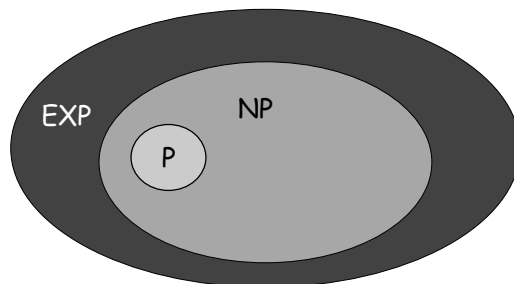Pf. Consider any problem X in NP.

- By definition, there exists a poly-time certifier C(s, t) for X.
- To solve input s, run C(s, t) on all strings t with $|t| \leq p(|s|)$.
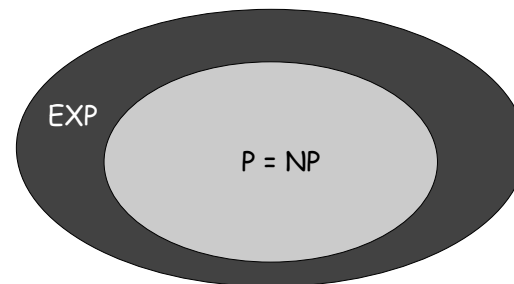- Return `yes`, if C(s, t) returns `yes` for any of these. ∎

The Main Question: P Versus NP

Does P = NP? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

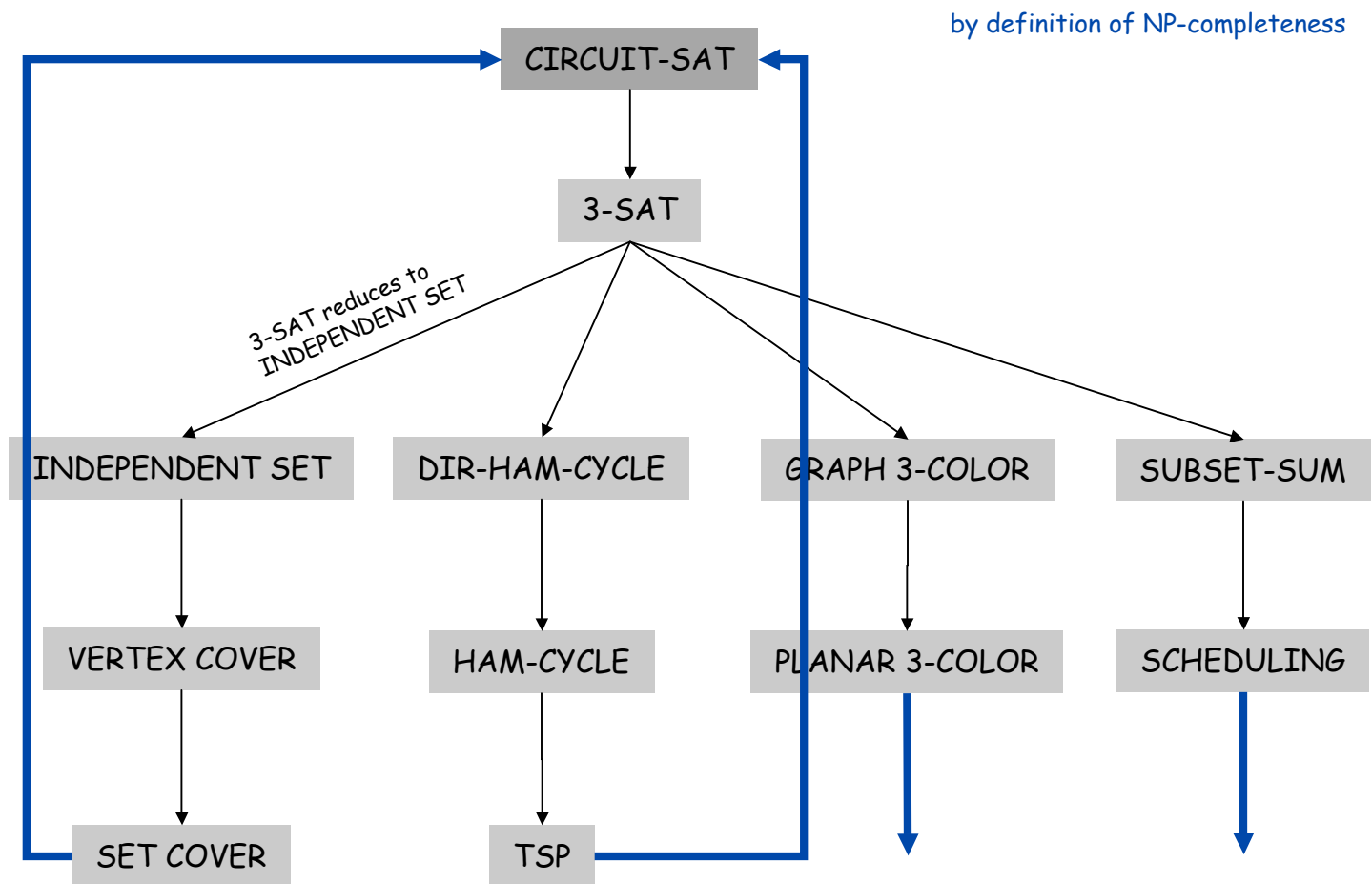- Is the decision problem as easy as the certification problem?
- Clay $1 million prize.

If P ≠ NP   If P = NP

NP-Completeness

Observation. All problems below are NP-complete and polynomial reduce to one another!

by definition of NP-completeness

CIRCUIT-SAT

3-SAT

3-SAT reduces to
INDEPENDENT SET

INDEPENDENT SET   DIR-HAM-CYCLE   GRAPH 3-COLOR   SUBSET-SUM

VERTEX COVER   HAM-CYCLE   PLANAR 3-COLOR   SCHEDULING

SET COVER   TSP

Some NP-Complete Problems

Six basic genres of NP-complete problems and paradigmatic examples.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

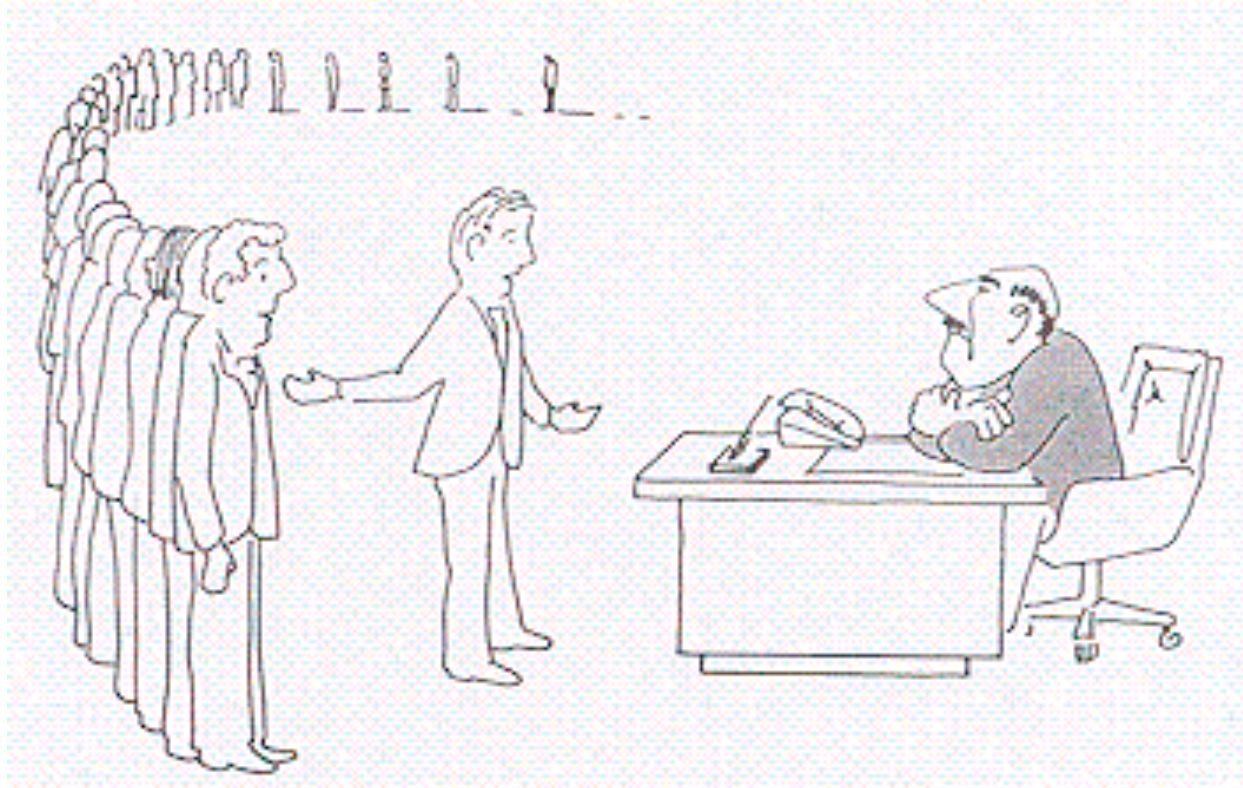Practice. Most NP problems are either known to be in P or NP-complete.

Notable exceptions. Factoring, graph isomorphism, Nash equilibrium.
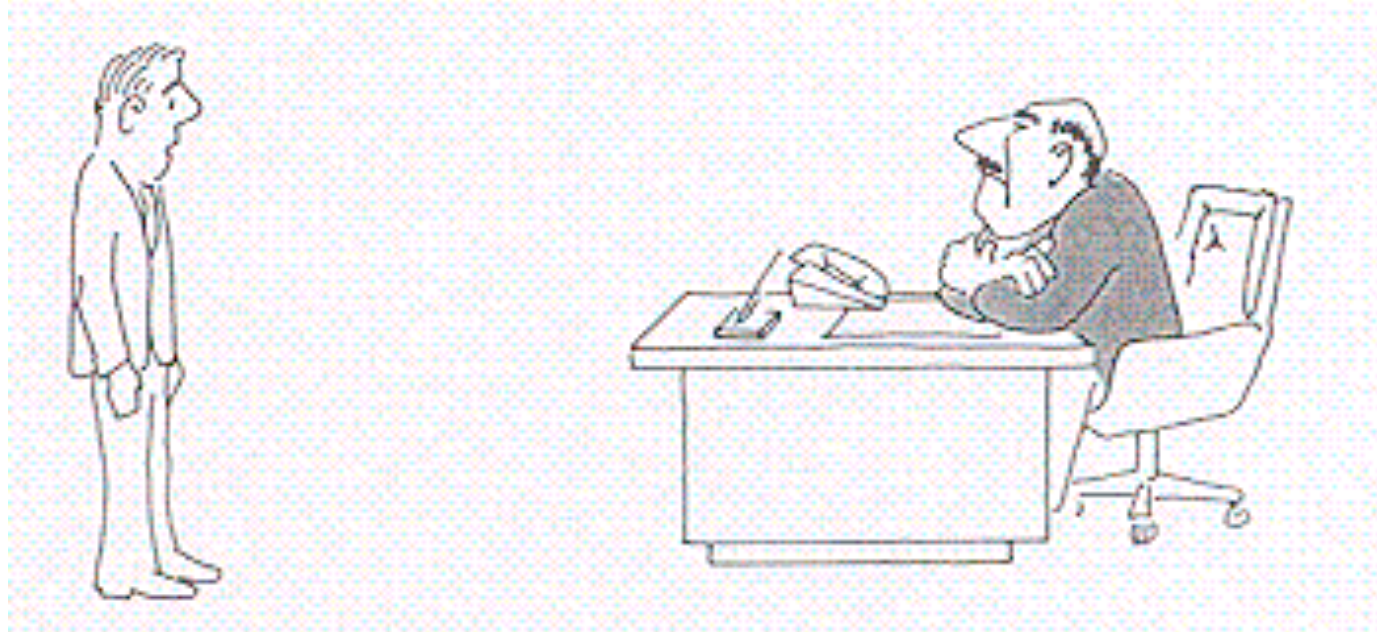
I can't find an efficient algorithm,
I guess I'm just too dumb.

I can't find an efficient algorithm,
because no such algorithm is possible.

I can't find an efficient algorithm, but neither can all these famous people.

WE MAY NOT BE ABLE TO SOLVE IT...
BUT WE SURE CAN GET CLOSE!

Approximation Algorithms

Q. Suppose I need to solve an NP-hard problem. What should I do?

A. Theory says you're unlikely to find a poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem to optimality.
- Solve problem in poly-time.
- Solve arbitrary instances of the problem.

ρ-approximation algorithm.
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio ρ of true optimum.

Challenge. Need to prove a solution's value is close to optimum, without even knowing what optimum value is!

11.1 Load Balancing

Load Balancing

Input. m identical machines; n jobs, job j has processing time t_j.
- Job j must run contiguously on one machine.
- A machine can process at most one job at a time.

Def. Let J(i) be the subset of jobs assigned to machine i. The load of machine i is $L_i = \Sigma_{j \in J(i)} t_j$.
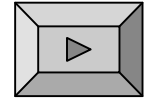
Def. The makespan is the maximum load on any machine $L = \max_i L_i$.

Load balancing. Assign each job to a machine to minimize makespan.

Load Balancing: List Scheduling

List-scheduling algorithm.

- Consider n jobs in some fixed order.
- Assign job j to machine whose load is smallest so far.

```
List-Scheduling(m, n, t₁,t₂,…,tₙ) {
    for i = 1 to m {
        Lᵢ ← 0          ←   load on machine i
        J(i) ← φ        ←   jobs assigned to machine i
    }

    for j = 1 to n {
        i = argminₖ Lₖ          ←   machine i has smallest load
        J(i) ← J(i) ∪ {j}       ←   assign job j to machine i
        Lᵢ ← Lᵢ + tⱼ            ←   update load of machine i
    }
}
```

Implementation. O(n log n) using a priority queue.

Load Balancing: List Scheduling
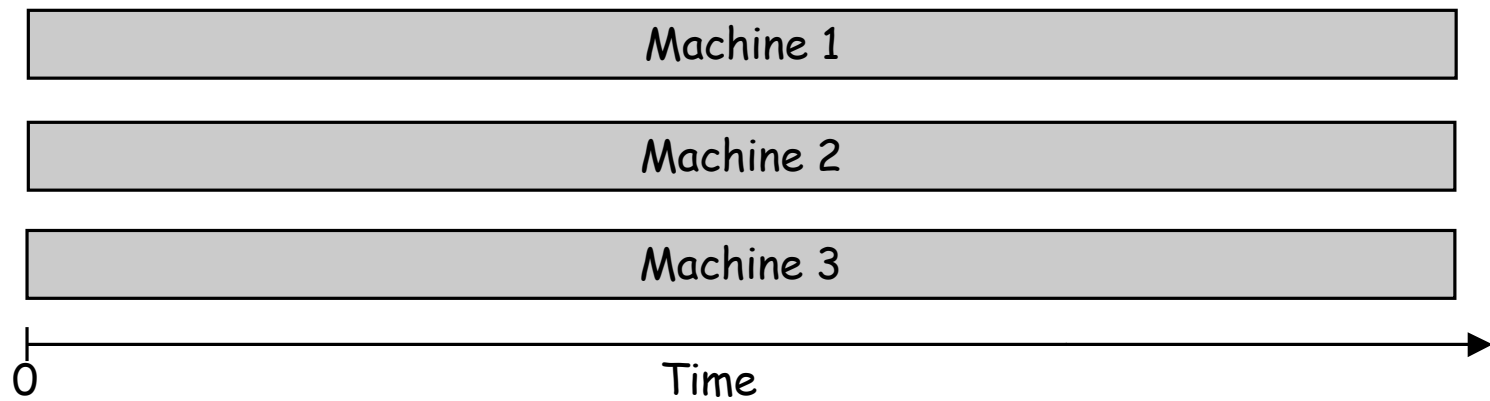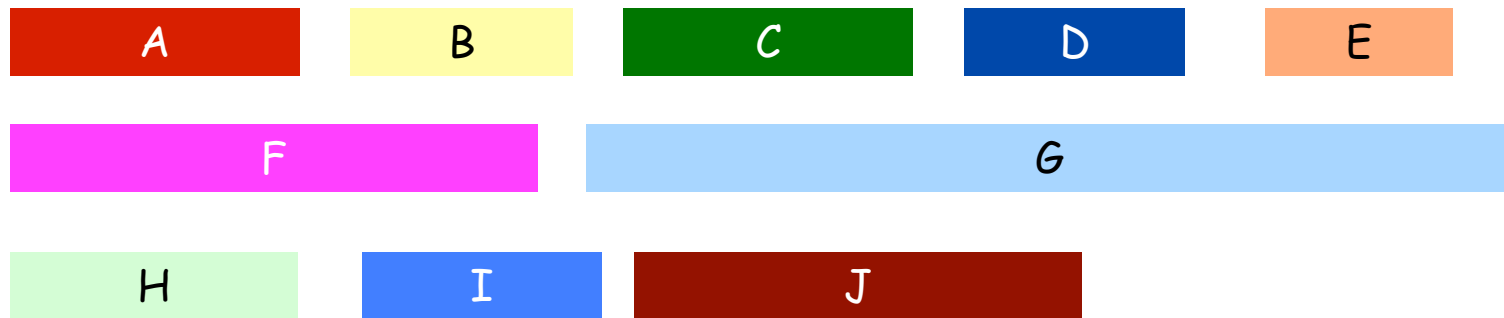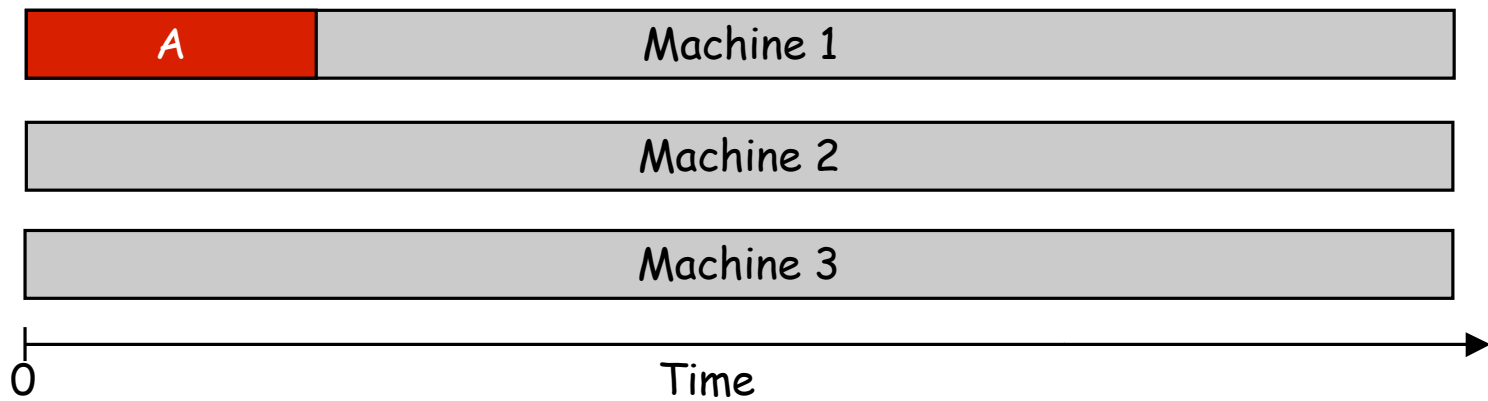
Load Balancing: List Scheduling

Load Balancing: List Scheduling

Load Balancing: List Scheduling

Load Balancing: List Scheduling

Load Balancing: List Scheduling

Load Balancing: List Scheduling

Load Balancing: List Scheduling
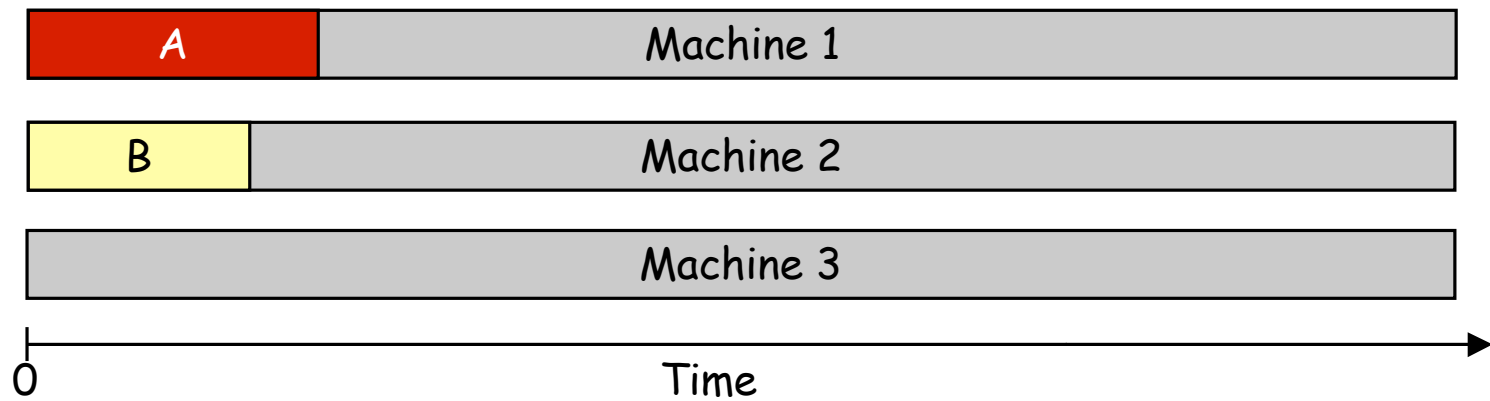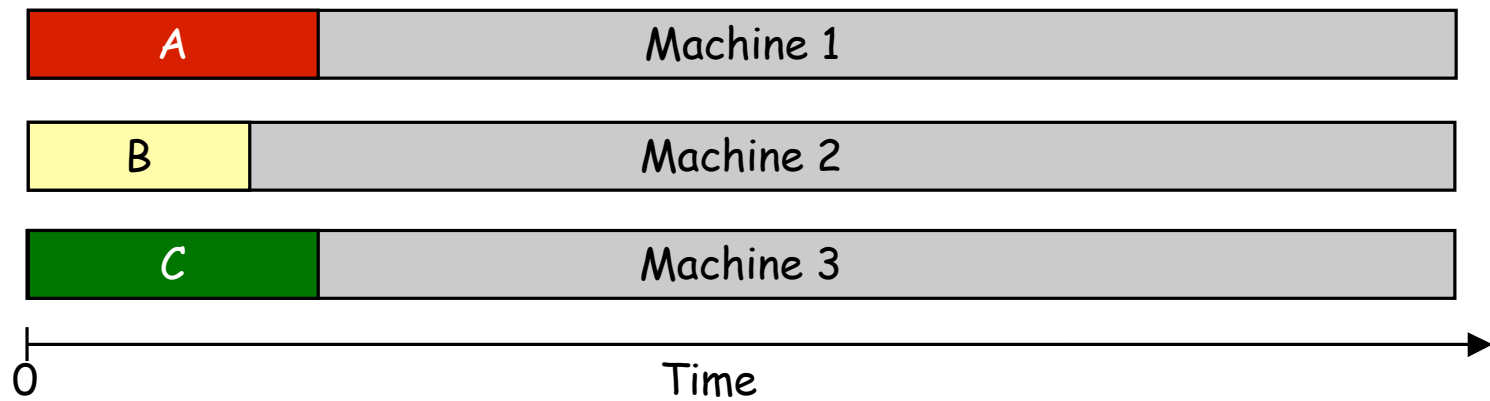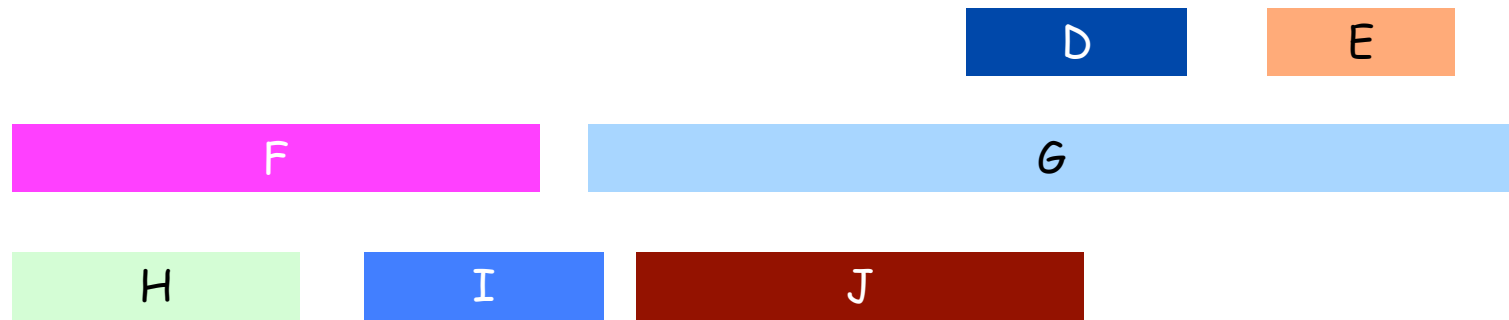
Load Balancing: List Scheduling

Load Balancing: List Scheduling

Load Balancing: List Scheduling

Load Balancing: List Scheduling

Optimal Schedule
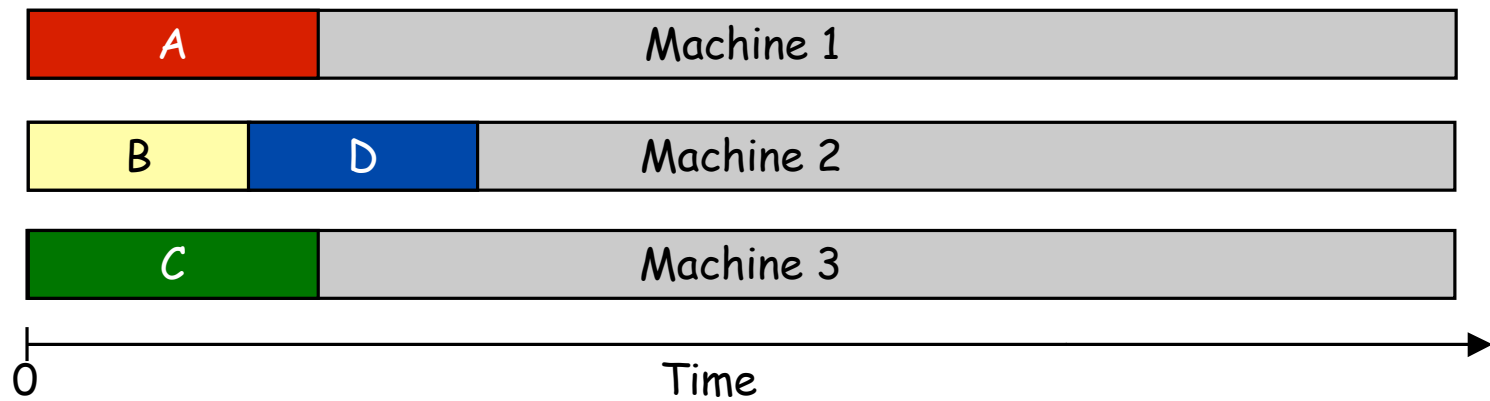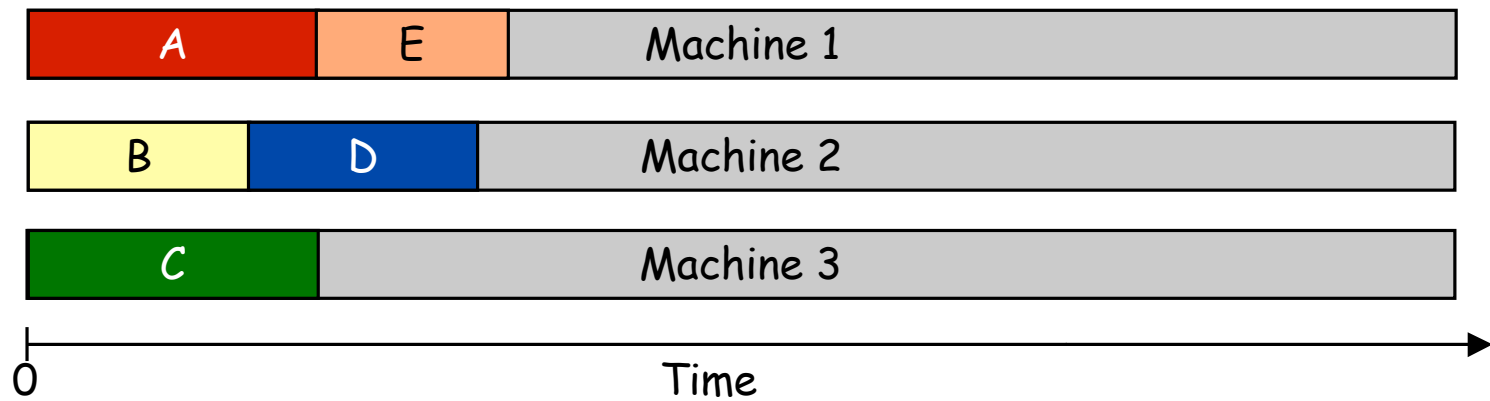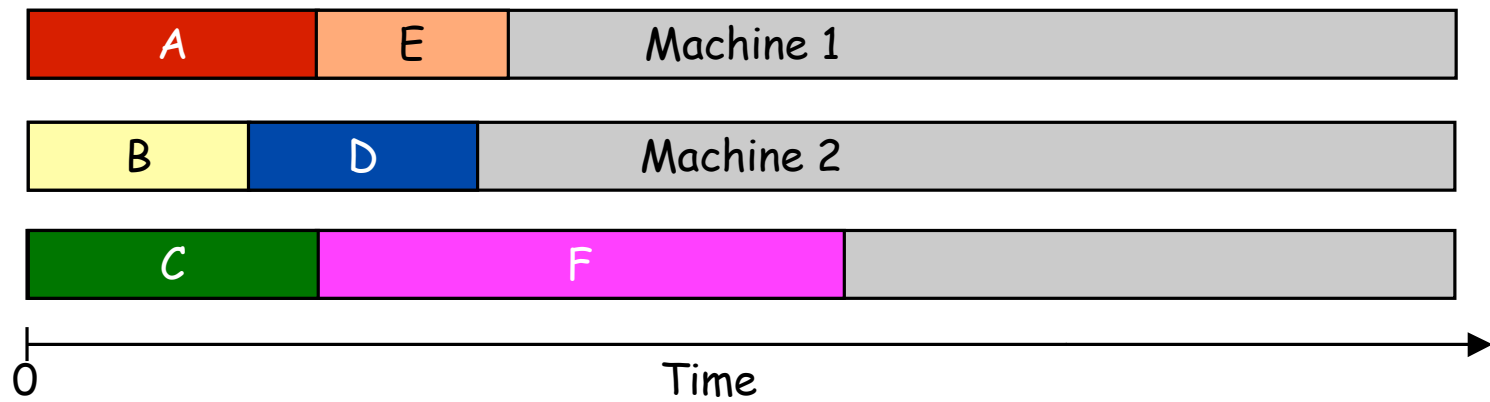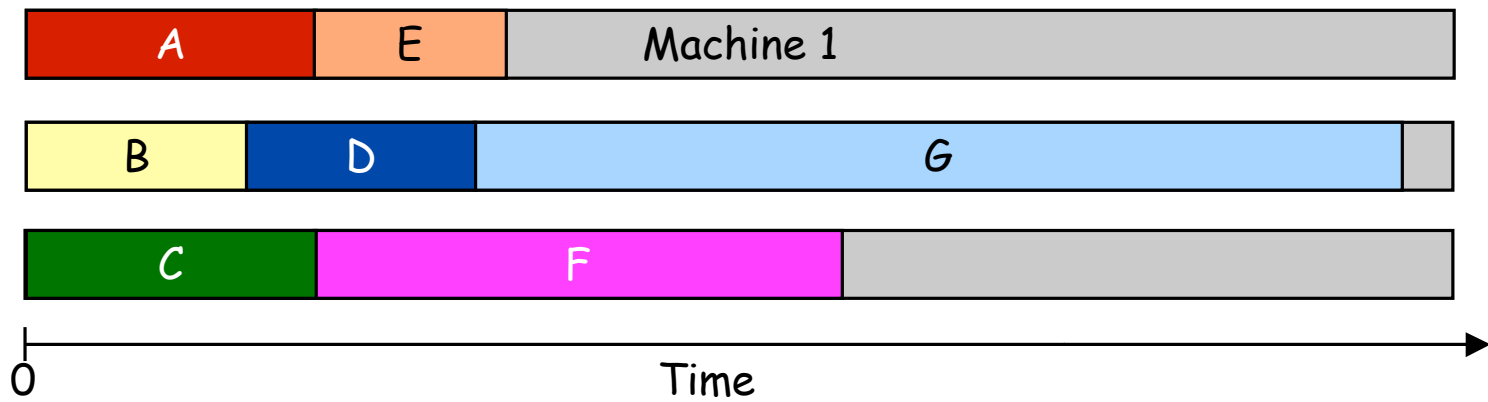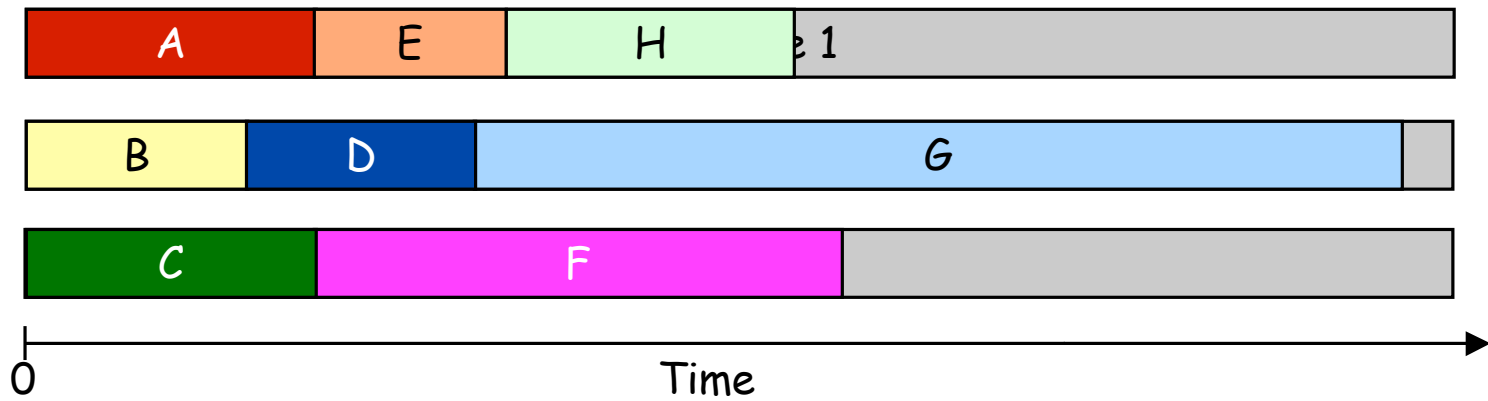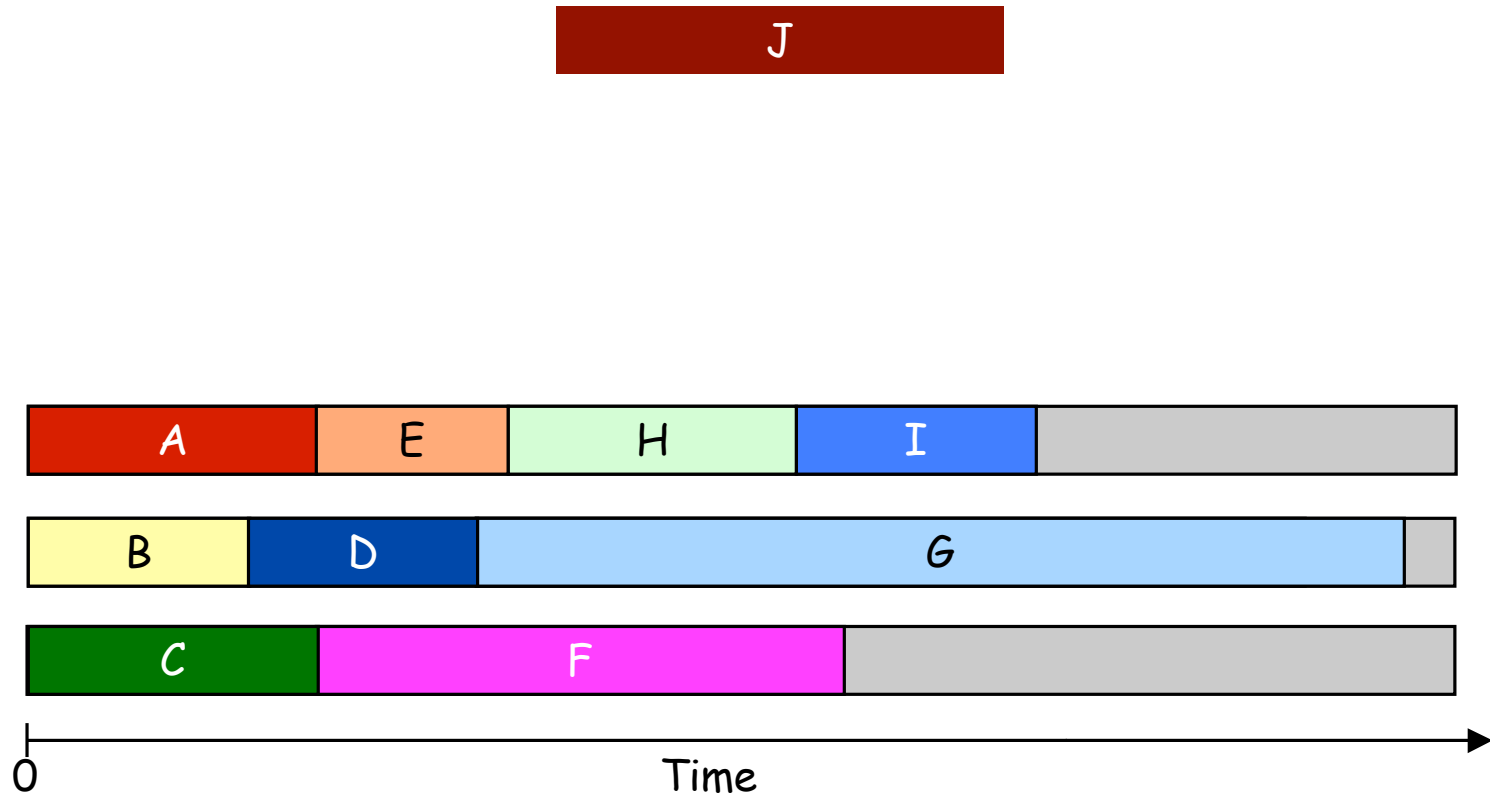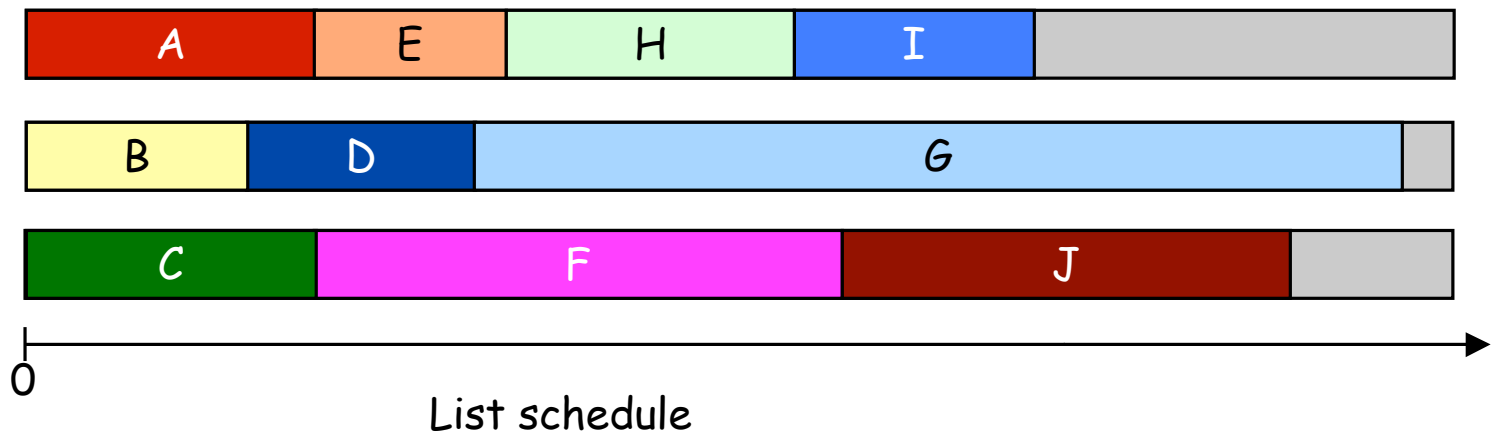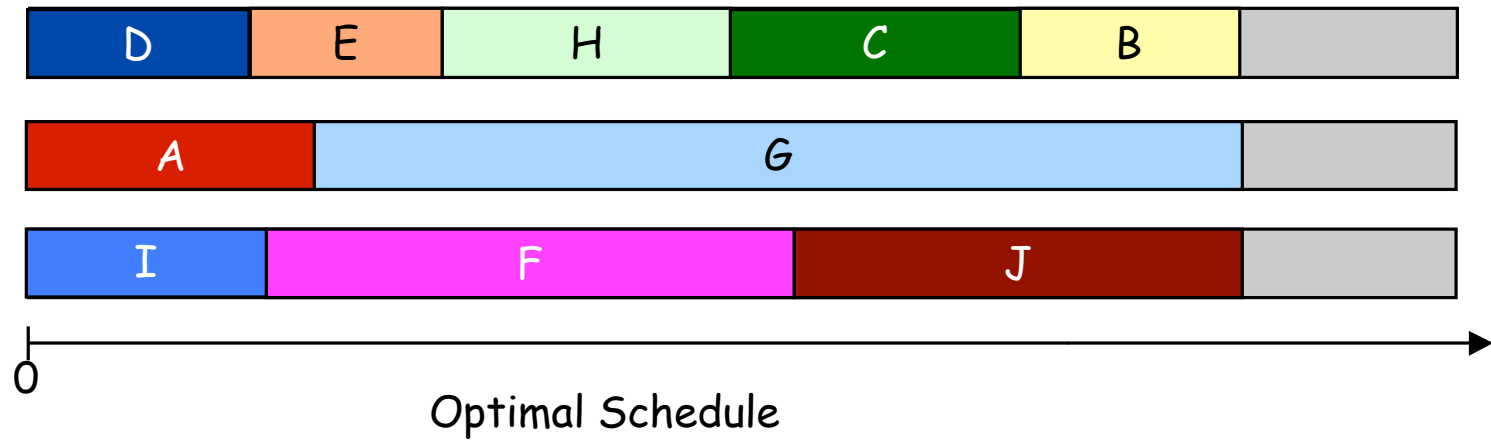
List schedule

Load Balancing: List Scheduling Analysis

Theorem. *[Graham, 1966]* Greedy algorithm is a 2-approximation.
- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan L*.

Lemma 1. The optimal makespan $L* \geq \max_j t_j$.

Pf. Some machine must process the most time-consuming job. ∎

Lemma 2. The optimal makespan $L* \geq \frac{1}{m}\sum_j t_j$.

Pf.
- The total processing time is $\sum_j t_j$.
- One of m machines must do at least a 1/m fraction of total work. ∎

Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load L_i of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j$ $\Rightarrow$ $L_i - t_j \leq L_k$ for all $1 \leq k \leq m$.

blue jobs scheduled before j

machine i

0

$L_i - t_j$

$L = L_i$
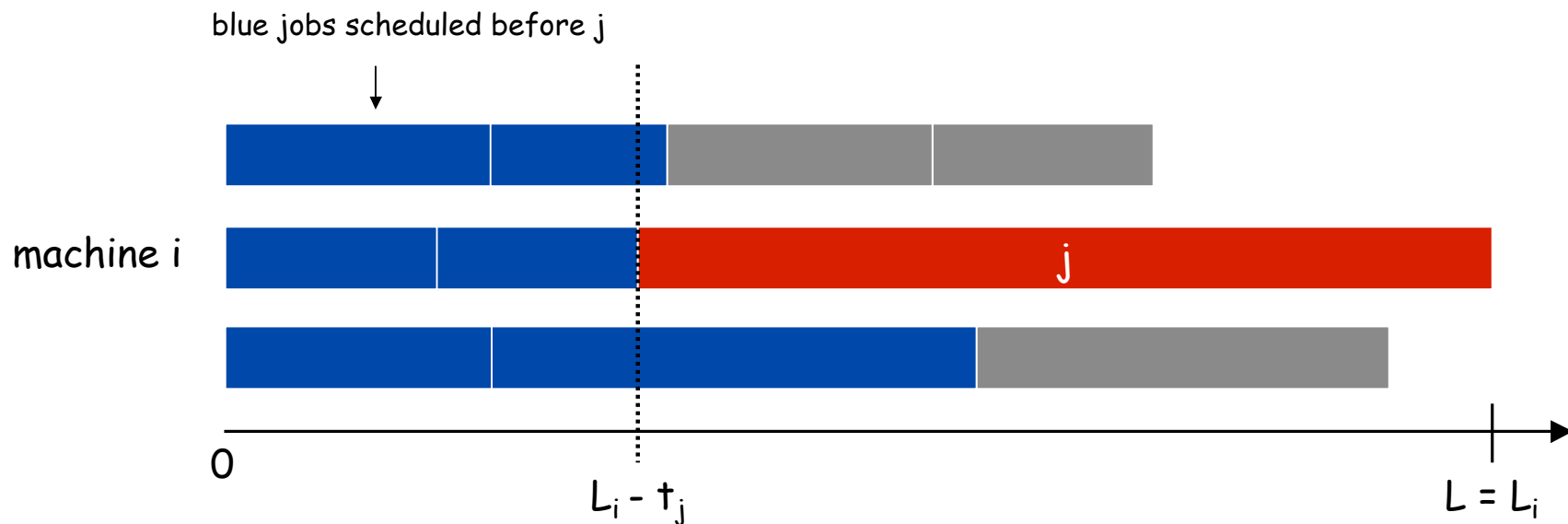
Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load L_i of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j$ $\Rightarrow$ $L_i - t_j \leq L_k$ for all $1 \leq k \leq m$.
- Sum inequalities over all k and divide by m:

$$
\begin{aligned}
L_i - t_j \ &\leq \ \tfrac{1}{m} \Sigma_k \, L_k \\
&= \ \tfrac{1}{m} \Sigma_k \, t_k \\
\text{Lemma 1} \longrightarrow \quad &\leq \ L*
\end{aligned}
$$

- Now $\quad L_i \ = \ \underbrace{(L_i - t_j)}_{\leq \, L*} \ + \ \underbrace{t_j}_{\leq \, L*} \ \leq \ 2L*.$ ∎

  $\uparrow$

  Lemma 2

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m

m = 10

								machine 2 idle	
								machine 3 idle	
								machine 4 idle	
								machine 5 idle	
								machine 6 idle	
								machine 7 idle	
								machine 8 idle	
								machine 9 idle	
								machine 10 idle	

list scheduling makespan = 19

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m

m = 10

optimal makespan = 10
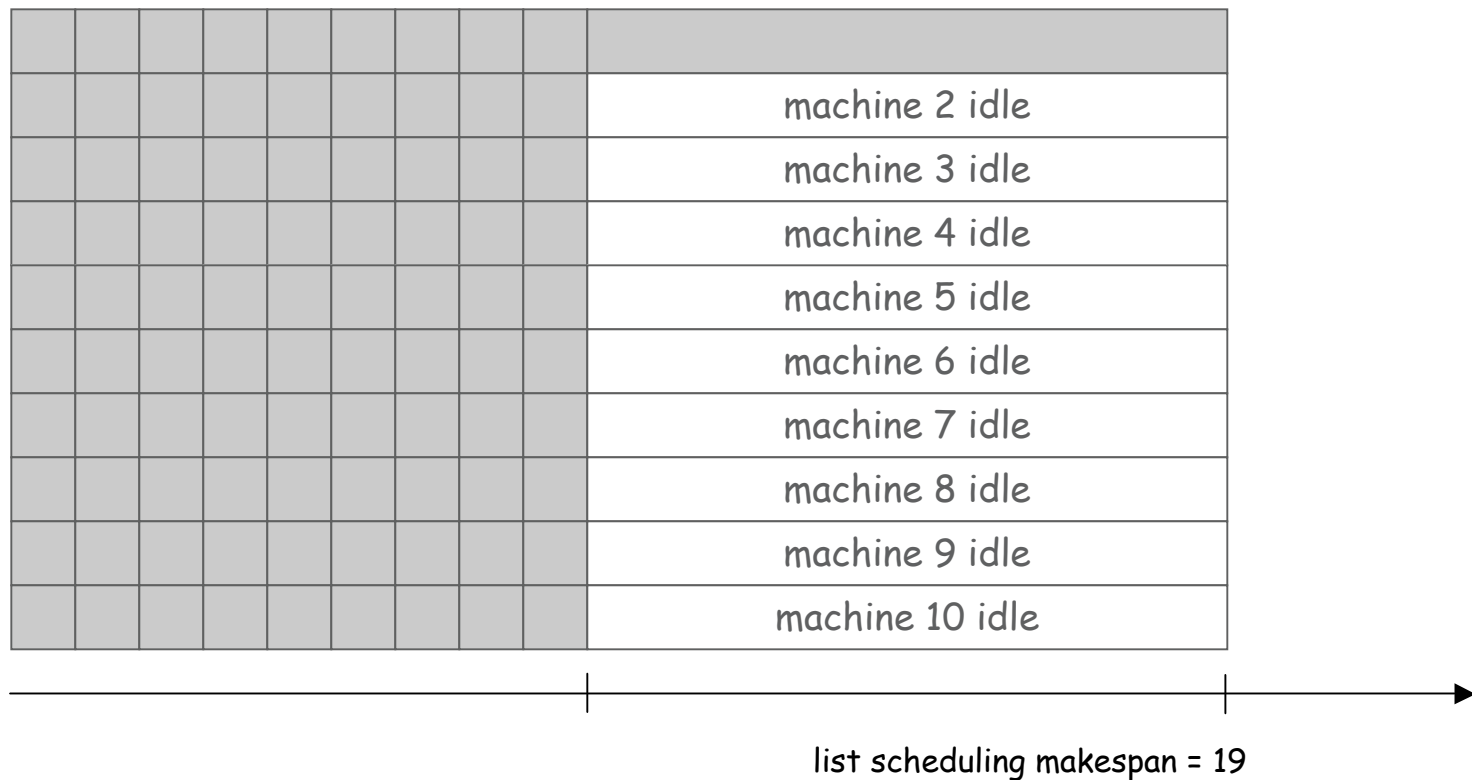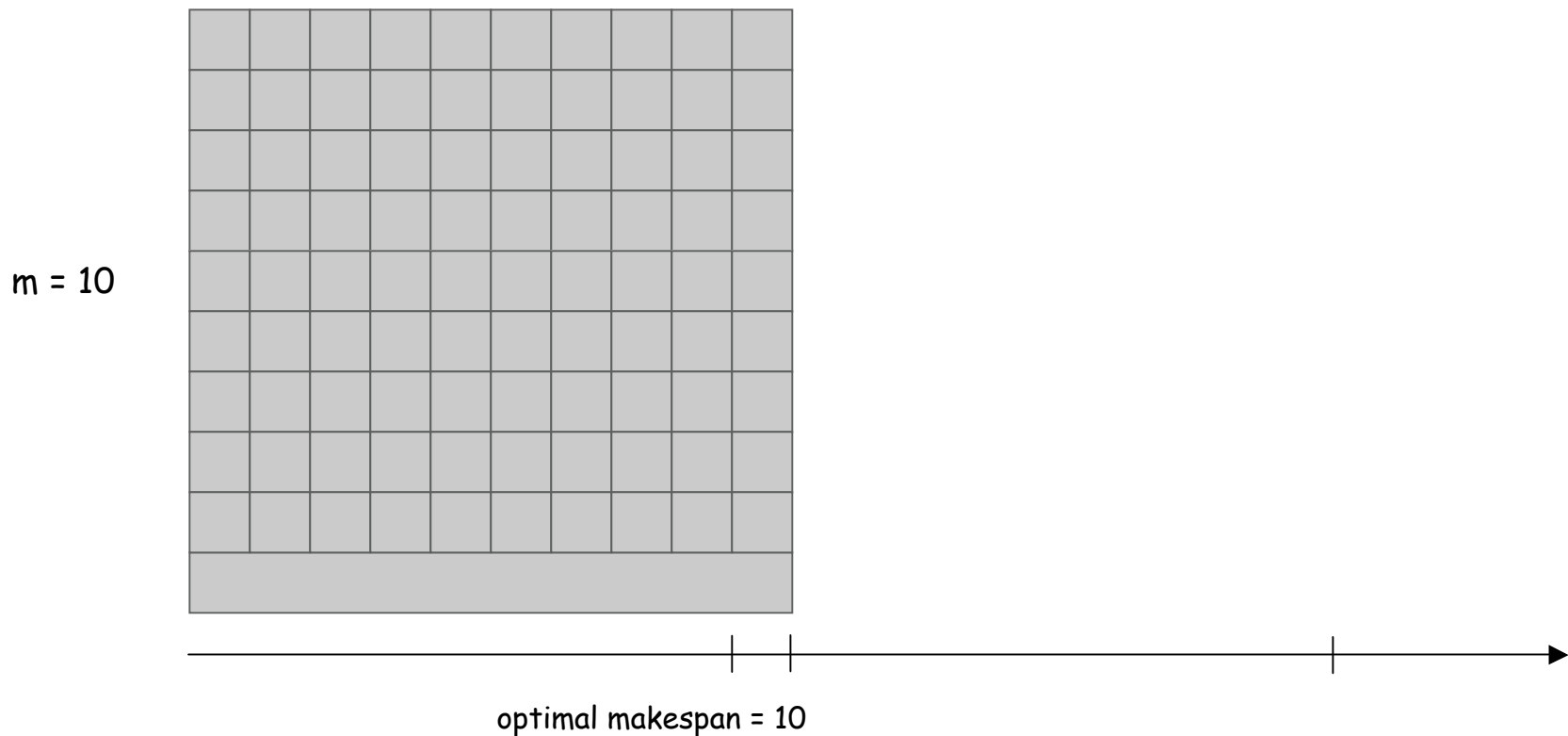
Load Balancing: LPT Rule

Longest processing time (LPT). Sort n jobs in descending order of processing time, and then run list scheduling algorithm.

```
LPT-List-Scheduling(m, n, t₁,t₂,…,tₙ) {
    Sort jobs so that t₁ ≥ t₂ ≥  … ≥ tₙ

    for i = 1 to m {
        Lᵢ ← 0          ←   load on machine i
        J(i) ← φ        ←   jobs assigned to machine i
    }


    for j = 1 to n {
        i = argminₖ Lₖ        ←   machine i has smallest load
        J(i) ← J(i) ∪ {j}     ←   assign job j to machine i
        Lᵢ ← Lᵢ + tⱼ           ←   update load of machine i
    }
}
```

Load Balancing: LPT Rule

Observation. If at most m jobs, then list-scheduling is optimal.

Pf. Each job put on its own machine. ∎

Lemma 3. If there are more than m jobs, $L^* \geq 2 t_{m+1}$.

Pf.

- Consider first m+1 jobs $t_1, \ldots, t_{m+1}$.
- Since the t_i's are in descending order, each takes at least t_{m+1} time.
- There are m+1 jobs and m machines, so by pigeonhole principle, at least one machine gets two jobs. ∎

Theorem. LPT rule is a 3/2 approximation algorithm.

Pf. Same basic approach as for list scheduling.

$$L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq \frac{1}{2}L^*} \leq \tfrac{3}{2}L^*. \quad \blacksquare$$

Lemma 3
(by observation, can assume number of jobs > m)

Load Balancing: LPT Rule

Q. Is our 3/2 analysis tight?
A. No.

Theorem. [Graham, 1969] LPT rule is a 4/3-approximation.
Pf. More sophisticated analysis of same algorithm.

Q. Is Graham's 4/3 analysis tight?
A. Essentially yes.

Ex: m machines, n = 2m+1 jobs, 2 jobs of length m+1, m+2, …, 2m–1 and three jobs of length m.

11.2 Center Selection

Center Selection Problem

Input. Set of n sites $s_1, \ldots, s_n$.

Center selection problem. Select k centers C so that maximum distance from a site to nearest center is minimized.
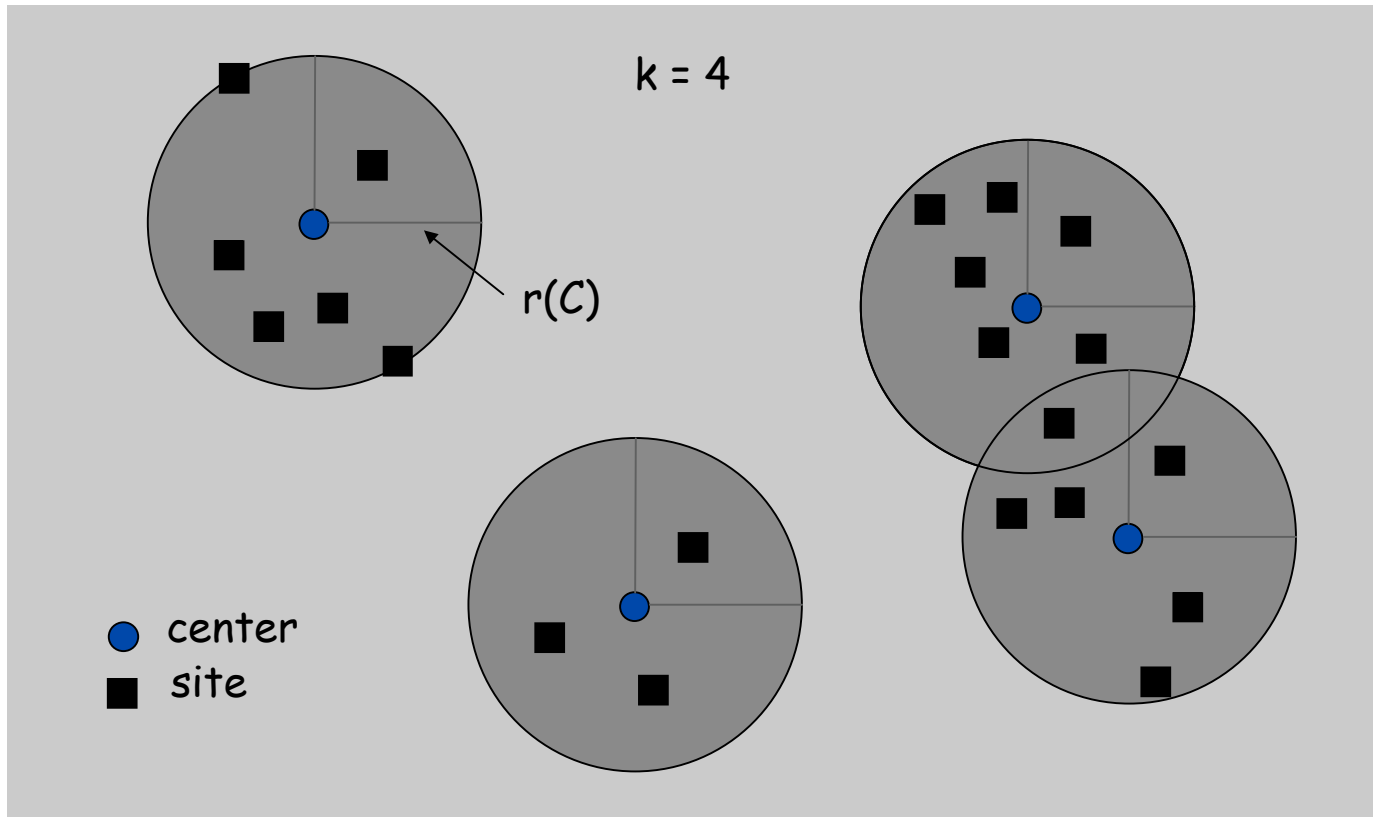
Center Selection Problem

Input. Set of n sites $s_1, ..., s_n$.

Center selection problem. Select k centers C so that maximum distance from a site to nearest center is minimized.

Notation.
- $dist(x, y)$ = distance between x and y.
- $dist(s_i, C) = \min_{c \in C} dist(s_i, c)$ = distance from s_i to closest center.
- $r(C) = \max_i dist(s_i, C)$ = smallest covering radius.

Goal. Find set of centers C that minimizes $r(C)$, subject to $|C| = k$.

Distance function properties.
- $dist(x, x) = 0$ (identity)
- $dist(x, y) = dist(y, x)$ (symmetry)
- $dist(x, y) \leq dist(x, z) + dist(z, y)$ (triangle inequality)

Center Selection Example

Ex: each site is a point in the plane, a center can be any point in the plane, dist(x, y) = Euclidean distance.

Remark: search can be infinite!

r(C)
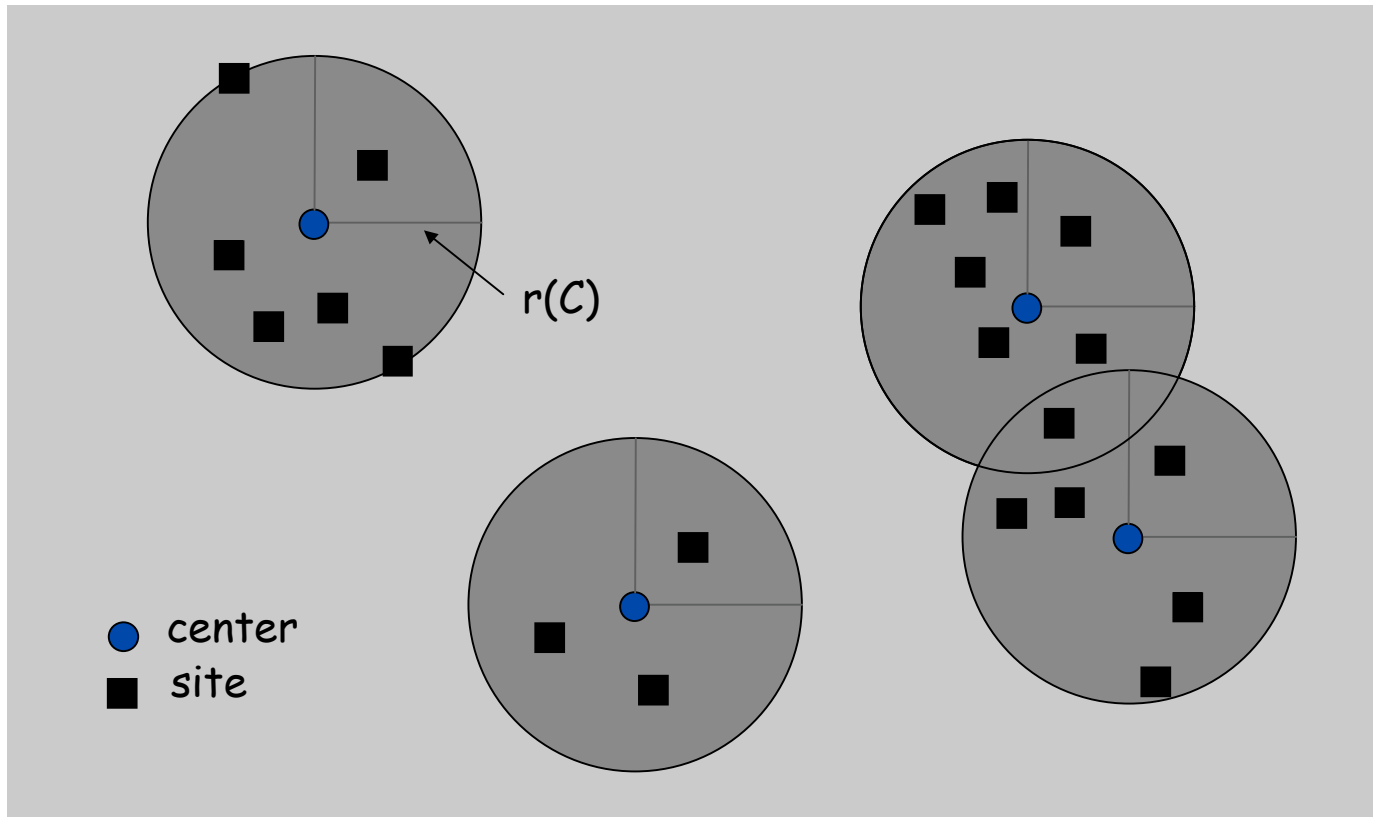
center
site

Greedy Algorithm: A False Start

Greedy algorithm. Put the first center at the best possible location for a single center, and then keep adding centers so as to reduce the covering radius each time by as much as possible.

Remark: arbitrarily bad!

greedy center 1

k = 2 centers
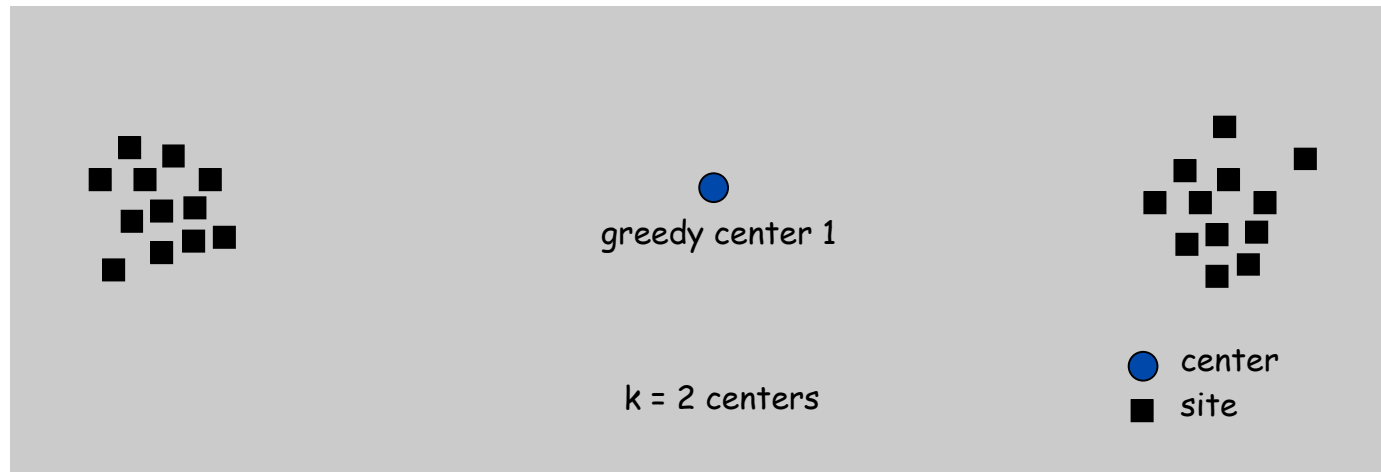
● center
■ site

Center Selection: Greedy Algorithm

Greedy algorithm. Repeatedly choose the next center to be the site
farthest from any existing center.

```
Greedy-Center-Selection(k, n, s₁,s₂,…,sₙ) {

    C = φ
    repeat k times {
        Select a site sᵢ with maximum dist(sᵢ, C)
        Add sᵢ to C                          ↑
    }                          site farthest from any center
    return C

}
```

Observation. Upon termination all centers in C are pairwise at least $r(C)$
apart.
Pf. By construction of algorithm.

Center Selection: Analysis of Greedy Algorithm

Theorem. Let C^* be an optimal set of centers. Then $r(C) \leq 2r(C^*)$.

Pf. (by contradiction) Assume $r(C^*) < \frac{1}{2} r(C)$.

- For each site c_i in C, consider ball of radius $\frac{1}{2} r(C)$ around it.
- Exactly one c_i^* in each ball; let c_i be the site paired with c_i^*.
- Consider any site s and its closest center c_i^* in C^*.
- $\text{dist}(s, C) \leq \text{dist}(s, c_i) \leq \text{dist}(s, c_i^*) + \text{dist}(c_i^*, c_i) \leq 2r(C^*)$.
- Thus $r(C) \leq 2r(C^*)$. ▪

Δ-inequality $\leq r(C^*)$ since c_i^* is closest center

$\frac{1}{2} r(C)$

$\frac{1}{2} r(C)$

$\frac{1}{2} r(C)$

c_i

c_i^*

s

● C^*
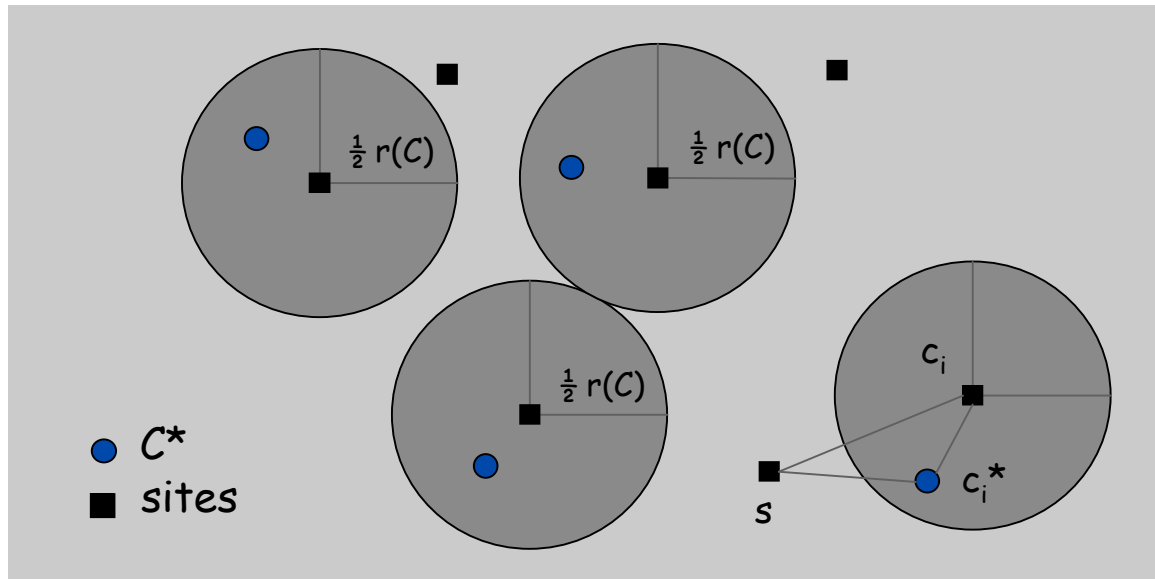
■ sites

Center Selection

Theorem. Let C^* be an optimal set of centers. Then $r(C) \leq 2r(C^*)$.

Theorem. Greedy algorithm is a 2-approximation for center selection problem.

Remark. Greedy algorithm always places centers at sites, but is still within a factor of 2 of best solution that is allowed to place centers anywhere.

e.g., points in the plane

Question. Is there hope of a 3/2-approximation? 4/3?

Theorem. Unless P = NP, there no ρ-approximation for center-selection problem for any $\rho < 2$.

11.4 The Pricing Method: Vertex Cover

Weighted Vertex Cover

Weighted vertex cover. Given a graph G with vertex weights, find a vertex cover of minimum weight.
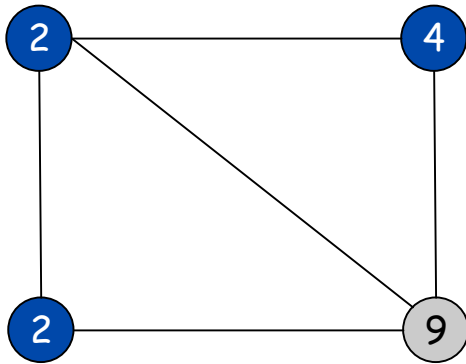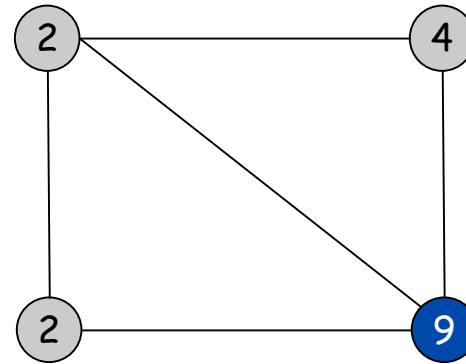
weight = 2 + 2 + 4   weight = 9

Weighted Vertex Cover

Pricing method. Each edge must be covered by some vertex i. Edge e pays price $p_e \geq 0$ to use vertex i.

Fairness. Edges incident to vertex i should pay $\leq w_i$ in total.

for each vertex i: $\displaystyle\sum_{e=(i,j)} p_e \leq w_i$
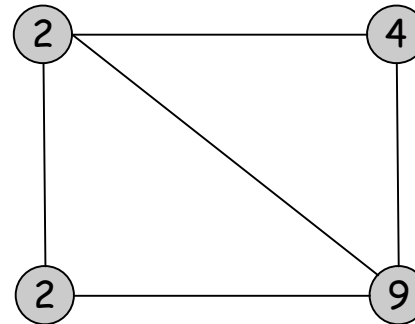
Claim. For any vertex cover S and any fair prices p_e: $\Sigma_e \, p_e \leq w(S)$.

Proof. ∎

$$\sum_{e \in E} p_e \;\leq\; \sum_{i \in S} \sum_{e=(i,j)} p_e \;\leq\; \sum_{i \in S} w_i \;=\; w(S).$$

↑

each edge e covered by
at least one node in S

↑

sum fairness inequalities
for each node in S

Pricing Method

Pricing method. Set prices and find vertex cover simultaneously.

```
Weighted-Vertex-Cover-Approx(G, w) {
    foreach e in E
        p_e = 0

    while (∃ edge i-j such that neither i nor j are tight)
        select such an edge e
        increase p_e without violating fairness
    }

    S ← set of all tight nodes
    return S
}
```

$$\sum_{e=(i,j)} p_e = w_i$$

Pricing Method
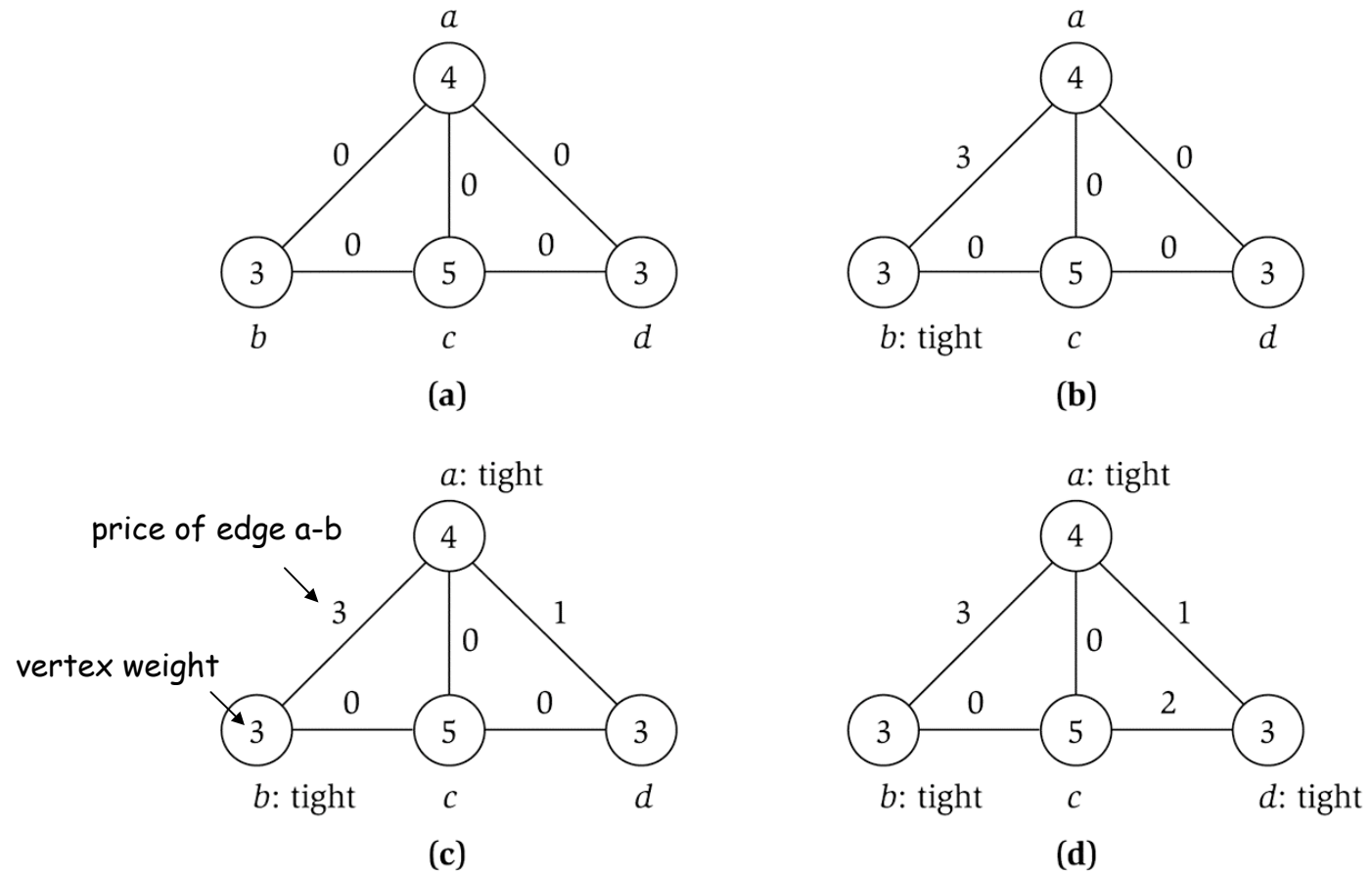
Figure 11.8

Pricing Method: Analysis

Theorem. Pricing method is a 2-approximation.
Pf.

- Algorithm terminates since at least one new node becomes tight after each iteration of while loop.

- Let S = set of all tight nodes upon termination of algorithm. S is a vertex cover: if some edge i-j is uncovered, then neither i nor j is tight. But then while loop would not terminate.

- Let S* be optimal vertex cover. We show $w(S) \leq 2w(S^*)$.

$$w(S) = \sum_{i \in S} w_i = \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in V} \sum_{e=(i,j)} p_e = 2 \sum_{e \in E} p_e \leq 2w(S^*). \quad \blacksquare$$

↑	↑	↑	↑
all nodes in S are tight	S ⊆ V, prices ≥ 0	each edge counted twice	fairness lemma

11.6 LP Rounding: Vertex Cover

Weighted Vertex Cover

Weighted vertex cover. Given an undirected graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a minimum weight subset of nodes S such that every edge is incident to at least one vertex in S.

total weight = 55
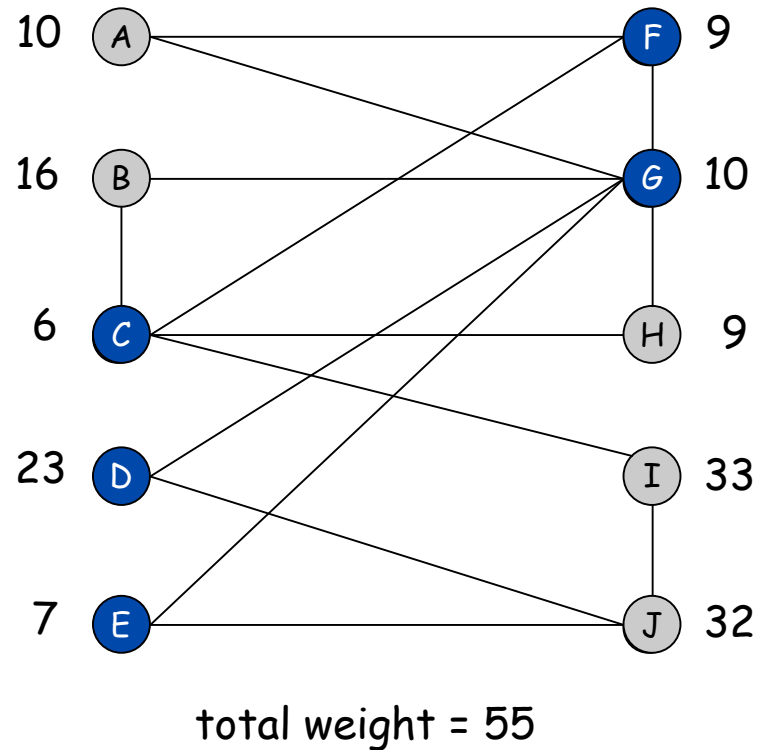
Weighted Vertex Cover: IP Formulation

Weighted vertex cover. Given an undirected graph G = (V, E) with vertex weights $w_i \geq 0$, find a minimum weight subset of nodes S such that every edge is incident to at least one vertex in S.

Integer programming formulation.
- Model inclusion of each vertex i using a 0/1 variable x_i.

$$x_i = \begin{cases} 0 & \text{if vertex } i \text{ is not in vertex cover} \\ 1 & \text{if vertex } i \text{ is in vertex cover} \end{cases}$$

  Vertex covers in 1-1 correspondence with 0/1 assignments:
  $S = \{i \in V : x_i = 1\}$

- Objective function: maximize $\Sigma_i \, w_i \, x_i$.

- Must take either i or j: $x_i + x_j \geq 1.$

Weighted Vertex Cover: IP Formulation

Weighted vertex cover. Integer programming formulation.

$$(ILP) \quad \min \quad \sum_{i \in V} w_i\, x_i$$
$$\text{s. t.} \quad x_i + x_j \quad \geq \quad 1 \qquad (i,j) \in E$$
$$x_i \qquad \in \quad \{0,1\} \quad i \in V$$

Observation. If x* is optimal solution to (ILP), then S = {i $\in$ V : x*$_i$ = 1} is a min weight vertex cover.

Integer Programming

INTEGER-PROGRAMMING. Given integers a_{ij} and b_i, find integers x_j that satisfy:

$$\max \quad c^t x$$
$$\text{s. t.} \quad Ax \geq b$$
$$x \quad \text{integral}$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad 1 \leq i \leq m$$
$$x_j \geq 0 \qquad 1 \leq j \leq n$$
$$x_j \quad \text{integral} \quad 1 \leq j \leq n$$

Observation. Vertex cover formulation proves that integer programming is NP-hard search problem.

even if all coefficients are 0/1 and
at most two variables per inequality

Linear Programming

Linear programming. Max/min linear objective function subject to linear inequalities.

- Input: integers c_j, b_i, a_{ij}.
- Output: real numbers x_j.

$$
\text{(P)} \quad \max \quad c^t x \\
\text{s. t.} \quad Ax \ \geq \ b \\
x \ \geq \ 0
$$

$$
\text{(P)} \quad \max \quad \sum_{j=1}^{n} c_j x_j \\
\text{s. t.} \quad \sum_{j=1}^{n} a_{ij} x_j \ \geq \ b_i \quad 1 \leq i \leq m \\
x_j \ \geq \ 0 \quad 1 \leq j \leq n
$$

Linear. No x^2, xy, $\arccos(x)$, $x(1\text{-}x)$, etc.

Simplex algorithm. [Dantzig 1947] Can solve LP in practice.
Ellipsoid algorithm. [Khachian 1979] Can solve LP in poly-time.

LP Feasible Region

LP geometry in 2D.

$x_1 = 0$

The region satisfying the inequalities
$$x_1 \geq 0, \; x_2 \geq 0$$
$$x_1 + \; 2x_2 \geq 6$$
$$2x_1 + \; x_2 \geq 6$$

$x_2 = 0$

$x_1 + 2x_2 = 6$

$2x_1 + x_2 = 6$
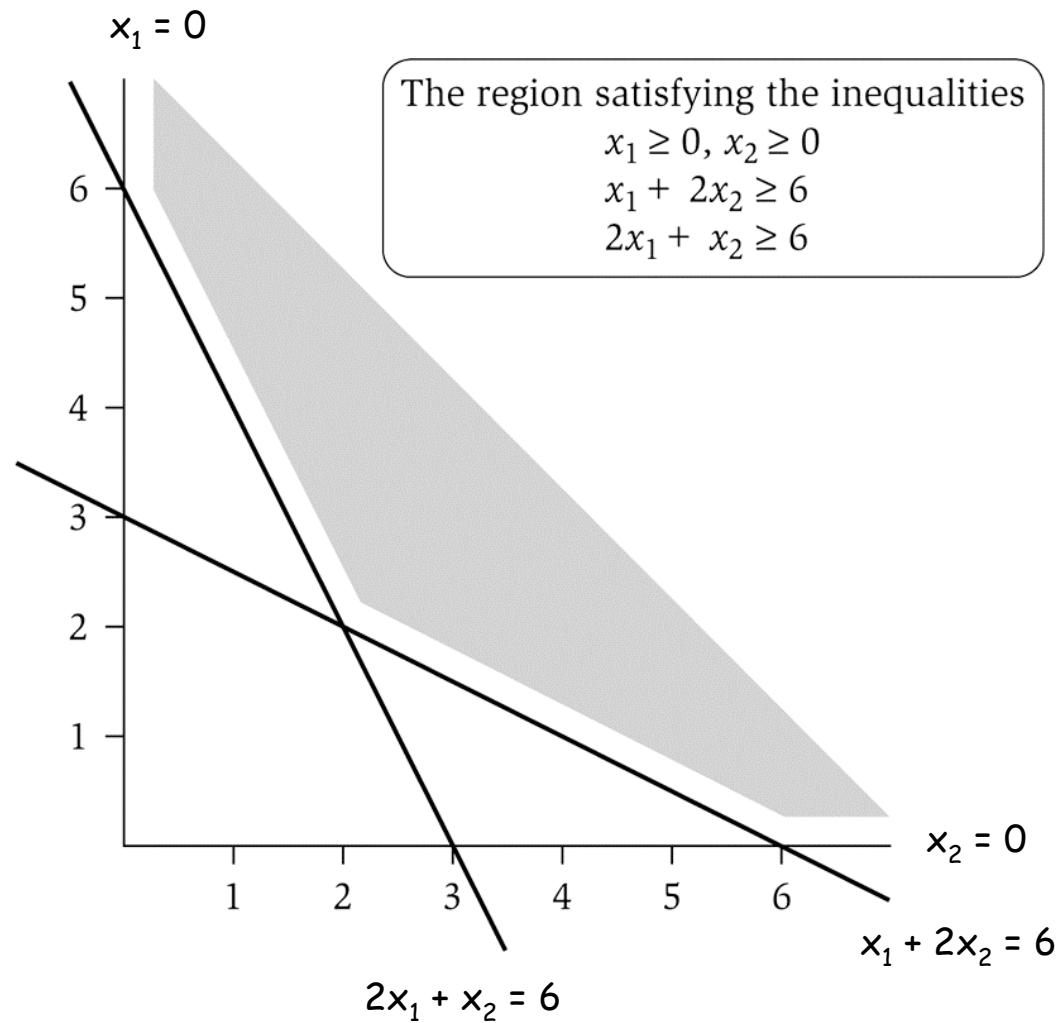
Weighted Vertex Cover: LP Relaxation

Weighted vertex cover. Linear programming formulation.

$$(LP) \quad \min \quad \sum_{i \, \in \, V} w_i \, x_i$$
$$\text{s.t.} \quad x_i + x_j \quad \geq \quad 1 \quad (i,j) \in E$$
$$x_i \qquad \quad \geq \quad 0 \quad i \in V$$

Observation. Optimal value of (LP) is ≤ optimal value of (ILP).

Pf. LP has fewer constraints.

Note. LP is not equivalent to vertex cover.

$\frac{1}{2}$ $\frac{1}{2}$

$\frac{1}{2}$
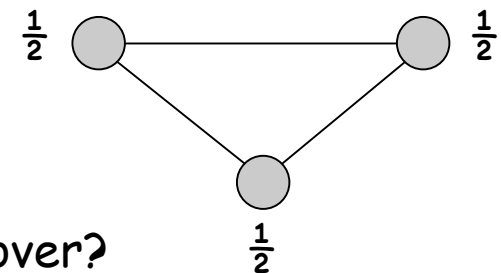
Q. How can solving LP help us find a small vertex cover?

A. Solve LP and round fractional values.

Weighted Vertex Cover

Theorem. If x^* is optimal solution to (LP), then $S = \{i \in V : x^*_i \geq \frac{1}{2}\}$ is a vertex cover whose weight is at most twice the min possible weight.

Pf. [S is a vertex cover]
- Consider an edge $(i, j) \in E$.
- Since $x^*_i + x^*_j \geq 1$, either $x^*_i \geq \frac{1}{2}$ or $x^*_j \geq \frac{1}{2}$ $\Rightarrow$ (i, j) covered.

Pf. [S has desired cost]
- Let S^* be optimal vertex cover. Then

$$\sum_{i \in S^*} w_i \;\geq\; \sum_{i \in S} w_i\, x_i^* \;\geq\; \frac{1}{2} \sum_{i \in S} w_i$$

$\uparrow$ $\uparrow$

LP is a relaxation $x^*_i \geq \frac{1}{2}$

Weighted Vertex Cover

Theorem. 2-approximation algorithm for weighted vertex cover.

Theorem. [Dinur-Safra 2001] If $P \neq NP$, then no ρ-approximation for $\rho < 1.3607$, even with unit weights.

$10 \sqrt{5} - 21$

Open research problem.   Close the gap.

* 11.7 Load Balancing Reloaded

Generalized Load Balancing

Input. Set of m machines M; set of n jobs J.

- Job j must run contiguously on an authorized machine in $M_j \subseteq M$.
- Job j has processing time t_j.
- Each machine can process at most one job at a time.

Def. Let J(i) be the subset of jobs assigned to machine i. The load of machine i is $L_i = \Sigma_{j \in J(i)} \, t_j$.

Def. The makespan is the maximum load on any machine = $\max_i L_i$.

Generalized load balancing. Assign each job to an authorized machine to minimize makespan.

Generalized Load Balancing: Integer Linear Program and Relaxation

ILP formulation. x_{ij} = time machine i spends processing job j.

$$
\begin{aligned}
(IP) \quad \min \quad & L \\
\text{s.t.} \quad & \sum_i x_{ij} = t_j && \text{for all } j \in J \\
& \sum_j x_{ij} \leq L && \text{for all } i \in M \\
& x_{ij} \in \{0, t_j\} && \text{for all } j \in J \text{ and } i \in M_j \\
& x_{ij} = 0 && \text{for all } j \in J \text{ and } i \notin M_j
\end{aligned}
$$

LP relaxation.

$$
\begin{aligned}
(LP) \quad \min \quad & L \\
\text{s.t.} \quad & \sum_i x_{ij} = t_j && \text{for all } j \in J \\
& \sum_j x_{ij} \leq L && \text{for all } i \in M \\
& x_{ij} \geq 0 && \text{for all } j \in J \text{ and } i \in M_j \\
& x_{ij} = 0 && \text{for all } j \in J \text{ and } i \notin M_j
\end{aligned}
$$

Generalized Load Balancing: Lower Bounds

Lemma 1. Let L be the optimal value to the LP. Then, the optimal makespan $L^* \geq L$.

Pf. LP has fewer constraints than IP formulation.

Lemma 2. The optimal makespan $L^* \geq \max_j t_j$.

Pf. Some machine must process the most time-consuming job. ▪

Generalized Load Balancing: Structure of LP Solution

Lemma 3. Let x be solution to LP. Let G(x) be the graph with an edge from machine i to job j if $x_{ij} > 0$. Then G(x) is acyclic.

Pf. (deferred)

can transform x into another LP solution where
G(x) is acyclic if LP solver doesn't return such an x

$x_{ij} > 0$

G(x) acyclic

G(x) cyclic

○ job
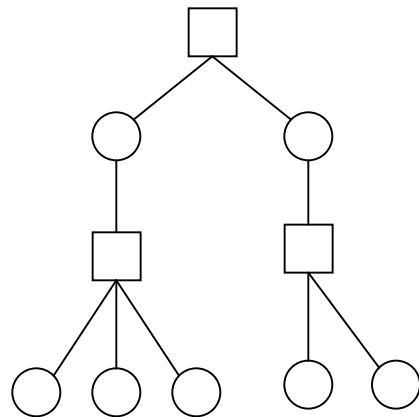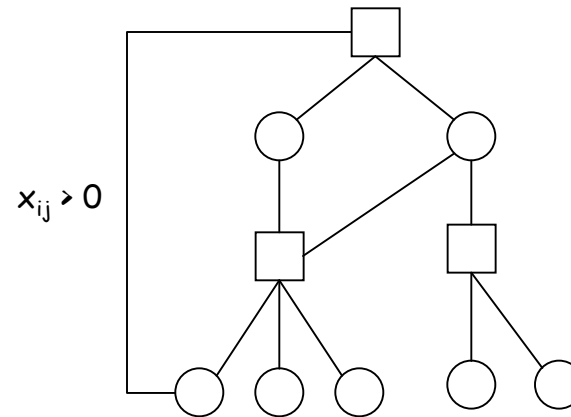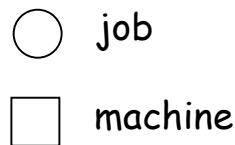
□ machine

Generalized Load Balancing: Rounding

Rounded solution. Find LP solution x where G(x) is a forest. Root forest G(x) at some arbitrary machine node r.

- If job j is a leaf node, assign j to its parent machine i.
- If job j is not a leaf node, assign j to one of its children.

Lemma 4. Rounded solution only assigns jobs to authorized machines.

Pf. If job j is assigned to machine i, then $x_{ij} > 0$. LP solution can only assign positive value to authorized machines. ∎

Each internal job node is assigned to an arbitrary child.

Each leaf is assigned to its parent.

○ job
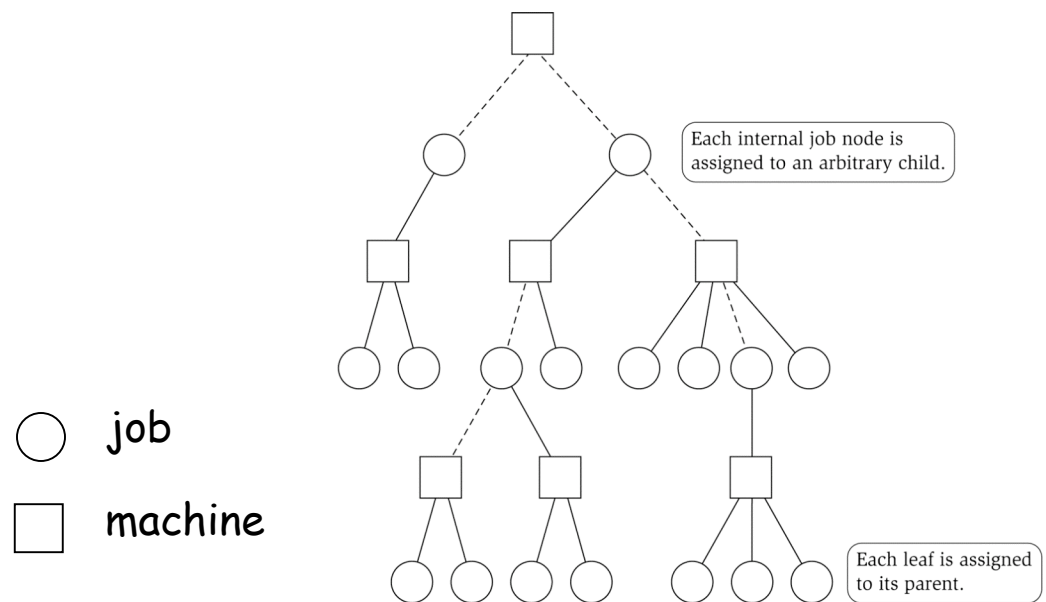
□ machine

Generalized Load Balancing: Analysis

Lemma 5. If job j is a leaf node and machine i = parent(j), then $x_{ij} = t_j$.
Pf. Since i is a leaf, $x_{ij} = 0$ for all $j \neq$ parent(i).   LP constraint
guarantees $\Sigma_i \, x_{ij} = t_j$.   ∎

Lemma 6. At most one non-leaf job is assigned to a machine.
Pf. The only possible non-leaf job assigned to machine i is parent(i).   ∎

○ job

□ machine

Each internal job node is assigned to an arbitrary child.

Each leaf is assigned to its parent.
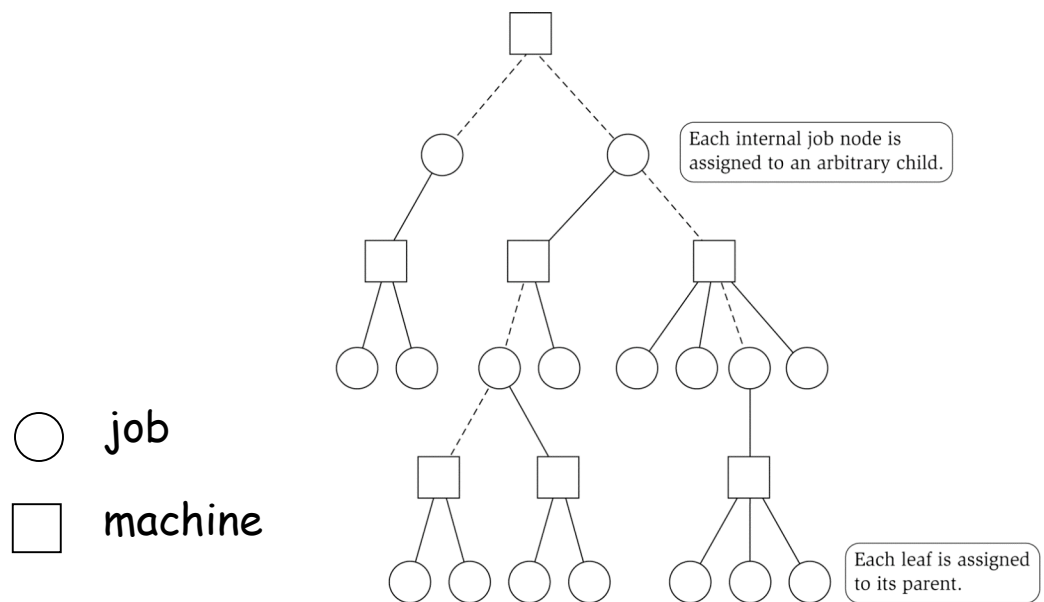
Generalized Load Balancing: Analysis

Theorem. Rounded solution is a 2-approximation.

Pf.

- Let $J(i)$ be the jobs assigned to machine i.

- By Lemma 6, the load L_i on machine i has two components:

  - leaf nodes

Lemma 5

LP   Lemma 1 (LP is a relaxation)

$$\sum_{\substack{j \in J(i) \\ j \text{ is a leaf}}} t_j \;=\; \sum_{\substack{j \in J(i) \\ j \text{ is a leaf}}} x_{ij} \;\leq\; \sum_{j \in J} x_{ij} \;\leq\; L \;\leq\; L^*$$

optimal value of LP

Lemma 2

  - parent(i) $t_{\text{parent}(i)} \;\leq\; L^*$

- Thus, the overall load $L_i \leq 2L^*$. ∎

Generalized Load Balancing: Flow Formulation

Flow formulation of LP.

$$\sum_i x_{ij} \;=\; t_j \quad \text{for all } j \in J$$

$$\sum_j x_{ij} \;\leq\; L \quad \text{for all } i \in M$$

$$x_{ij} \;\geq\; 0 \quad \text{for all } j \in J \text{ and } i \in M_j$$

$$x_{ij} \;=\; 0 \quad \text{for all } j \in J \text{ and } i \notin M_j$$

Jobs

∞ Machines

Supply = t_j j

L

i

L

v Demand = $\sum_j t_j$

L

L
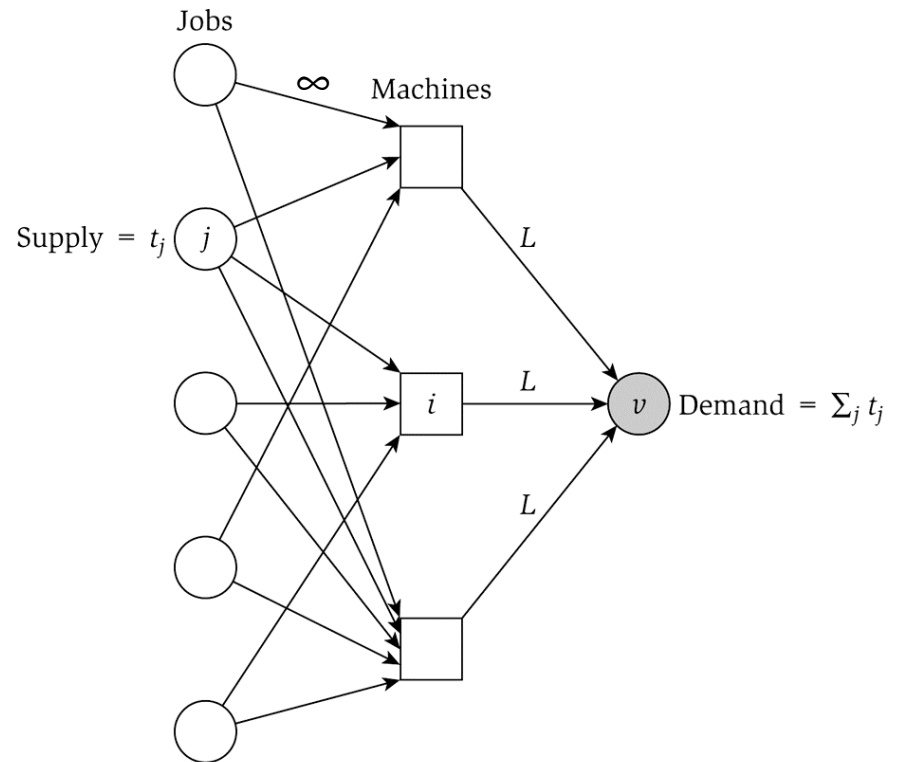
Observation. Solution to feasible flow problem with value L are in one-to-one correspondence with LP solutions of value L.

Generalized Load Balancing: Structure of Solution

Lemma 3. Let (x, L) be solution to LP. Let $G(x)$ be the graph with an edge from machine i to job j if $x_{ij} > 0$. We can find another solution (x', L) such that $G(x')$ is acyclic.

Pf. Let C be a cycle in $G(x)$.
- Augment flow along the cycle C. $\longleftarrow$ flow conservation maintained
- At least one edge from C is removed (and none are added).
- Repeat until $G(x')$ is acyclic.

$G(x)$

augment along C

$G(x')$
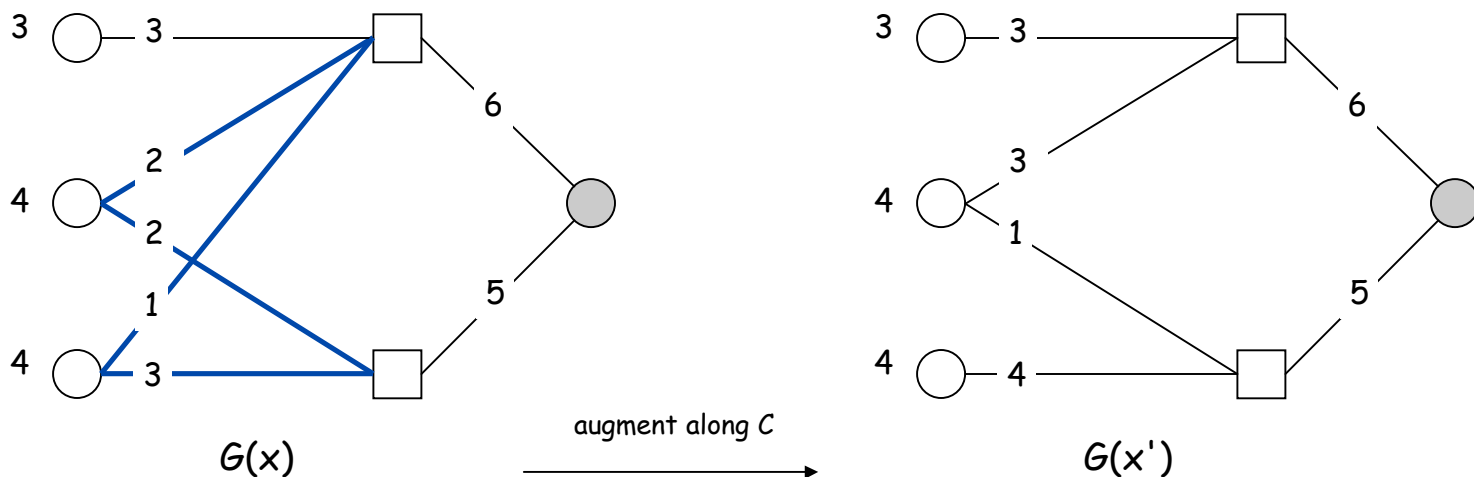
Conclusions

Running time. The bottleneck operation in our 2-approximation is solving one LP with $mn + 1$ variables.

Remark. Can solve LP using flow techniques on a graph with $m+n+1$ nodes: given L, find feasible flow if it exists. Binary search to find L*.

Extensions: unrelated parallel machines. [Lenstra-Shmoys-Tardos 1990]
- Job j takes t_{ij} time if processed on machine i.
- 2-approximation algorithm via LP rounding.
- No 3/2-approximation algorithm unless P = NP.

11.8 Knapsack Problem

Polynomial Time Approximation Scheme

PTAS. $(1 + \varepsilon)$-approximation algorithm for any constant $\varepsilon > 0$.
- Load balancing. [Hochbaum-Shmoys 1987]
- Euclidean TSP. [Arora 1996]

Consequence. PTAS produces arbitrarily high quality solution, but trades off accuracy for time.

This section. PTAS for knapsack problem via rounding and scaling.

Knapsack Problem

Knapsack problem.

- Given n objects and a "knapsack."
- Item i has value $v_i > 0$ and weighs $w_i > 0$. ⟵ we'll assume $w_i \leq W$
- Knapsack can carry weight up to W.
- Goal: fill knapsack so as to maximize total value.

Ex: { 3, 4 } has value 40.

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Knapsack is NP-Complete

KNAPSACK: Given a finite set X, nonnegative weights w_i, nonnegative values v_i, a weight limit W, and a target value V, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} w_i \leq W$$

$$\sum_{i \in S} v_i \geq V$$

SUBSET-SUM: Given a finite set X, nonnegative values u_i, and an integer U, is there a subset $S \subseteq X$ whose elements sum to exactly U?

Claim. SUBSET-SUM $\leq_P$ KNAPSACK.

Pf. Given instance $(u_1, ..., u_n, U)$ of SUBSET-SUM, create KNAPSACK instance:

$$v_i = w_i = u_i \qquad \sum_{i \in S} u_i \leq U$$

$$V = W = U \qquad \sum_{i \in S} u_i \geq U$$

Knapsack Problem: Dynamic Programming 1

Def. OPT(i, w) = max value subset of items 1,..., i with weight limit w.

- Case 1: OPT does not select item i.
  - OPT selects best of 1, ..., i−1 using up to weight limit w
- Case 2: OPT selects item i.
  - new weight limit = w − w_i
  - OPT selects best of 1, ..., i−1 using up to weight limit w − w_i

$$
OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w), \ v_i + OPT(i-1, w-w_i)\} & \text{otherwise} \end{cases}
$$

Running time. O(n W).

- W = weight limit.
- Not polynomial in input size!

Knapsack Problem: Dynamic Programming II

Def. OPT(i, v) = min weight subset of items 1, …, i that yields value exactly v.

- Case 1: OPT does not select item i.
  - OPT selects best of 1, …, i-1 that achieves exactly value v
- Case 2: OPT selects item i.
  - consumes weight w_i, new value needed = $v - v_i$
  - OPT selects best of 1, …, i-1 that achieves exactly value v

$$OPT(i,v) = \begin{cases} 0 & \text{if } v = 0 \\ \infty & \text{if } i = 0, v > 0 \\ OPT(i-1, v) & \text{if } v_i > v \\ \min\{OPT(i-1, v), \ w_i + OPT(i-1, v-v_i)\} & \text{otherwise} \end{cases}$$

$V^* \leq n \ v_{max}$

Running time. $O(n \ V^*) = O(n^2 \ v_{max})$.

- V^* = optimal value = maximum v such that $OPT(n, v) \leq W$.
- Not polynomial in input size!

Knapsack: FPTAS

Intuition for approximation algorithm.

- Round all values up to lie in smaller range.
- Run dynamic programming algorithm on rounded instance.
- Return optimal items in rounded instance.

Item	Value	Weight
1	134,221	1
2	656,342	2
3	1,810,013	5
4	22,217,800	6
5	28,343,199	7

W = 11

original instance

Item	Value	Weight
1	2	1
2	7	2
3	19	5
4	23	6
5	29	7

W = 11

rounded instance

Knapsack: FPTAS

Knapsack FPTAS. Round up all values: $\quad \overline{v}_i = \left\lceil \dfrac{v_i}{\theta} \right\rceil \theta, \quad \hat{v}_i = \left\lceil \dfrac{v_i}{\theta} \right\rceil$

- v_{max} = largest value in original instance
- ε = precision parameter
- θ = scaling factor = $\varepsilon \, v_{max} \, / \, n$

Observation. Optimal solution to problems with $\overline{v}$ or $\hat{v}$ are equivalent.

Intuition. $\overline{v}$ close to v so optimal solution using $\overline{v}$ is nearly optimal; $\hat{v}$ small and integral so dynamic programming algorithm is fast.

Running time. $O(n^3 / \varepsilon)$.

- Dynamic program II running time is $O(n^2 \, \hat{v}_{max})$, where

$$\hat{v}_{max} = \left\lceil \frac{v_{max}}{\theta} \right\rceil = \left\lceil \frac{n}{\varepsilon} \right\rceil$$

Knapsack: FPTAS

Knapsack FPTAS. Round up all values: $\quad \bar{v}_i = \left\lceil \dfrac{v_i}{\theta} \right\rceil \theta$

Theorem. If S is solution found by our algorithm and S* is any other feasible solution then $\quad (1+\varepsilon) \sum\limits_{i \in S} v_i \;\geq\; \sum\limits_{i \in S^*} v_i$

Pf. Let S* be any feasible solution satisfying weight constraint.

$$\sum_{i \in S^*} v_i \;\leq\; \sum_{i \in S^*} \bar{v}_i \qquad\qquad \text{always round up}$$

$$\leq\; \sum_{i \in S} \bar{v}_i \qquad\qquad \text{solve rounded instance optimally}$$

$$\leq\; \sum_{i \in S} (v_i + \theta) \qquad\qquad \text{never round up by more than } \theta$$

$$\leq\; \sum_{i \in S} v_i + n\theta \qquad\qquad |S| \leq n$$

DP alg can take v_{max}

$$\leq\; (1+\varepsilon) \sum_{i \in S} v_i \qquad\qquad n\,\theta = \varepsilon\, v_{max}, \; v_{max} \leq \Sigma_{i \in S}\, v_i$$

Extra Slides

Load Balancing on 2 Machines

Claim. Load balancing is hard even if only 2 machines.
Pf. NUMBER-PARTITIONING $\leq_P$ LOAD-BALANCE.

NP-complete by Exercise 8.26

length of job f

machine 1

yes

machine 2

0   Time   L

Center Selection: Hardness of Approximation

Theorem. Unless P = NP, there is no ρ-approximation algorithm for metric k-center problem for any $\rho < 2$.

Pf. We show how we could use a $(2 - \varepsilon)$ approximation algorithm for k-center to solve DOMINATING-SET in poly-time.

- Let G = (V, E), k be an instance of DOMINATING-SET. ⟵ *see Exercise 8.29*
- Construct instance G' of k-center with sites V and distances
  - d(u, v) = 2 if (u, v) $\in$ E
  - d(u, v) = 1 if (u, v) $\notin$ E
- Note that G' satisfies the triangle inequality.
- Claim: G has dominating set of size k iff there exists k centers C* with r(C*) = 1.
- Thus, if G has a dominating set of size k, a $(2 - \varepsilon)$-approximation algorithm on G' must find a solution C* with r(C*) = 1 since it cannot use any edge of distance 2.